Ph.D Thesis

Development of PCI embedded card useful for microcontroller trainer

By

Chirutkar Harshadkumar Govindrao

Under the guidance of

Dr. H. N. Pandya

Prof. & Head

Department of Electronics,

Saurashtra University,

Rajkot - 360005

Gujarat (India)

May – 2011

DEVELOPMENT OF PCI EMBEDDED CARD USEFUL FOR MICROCONTROLLER TRAINER

The Thesis

Submitted to Saurashtra University, Rajkot

For

The Degree of Doctor of Philosophy

In

The Faculty of Science

In

The Subject of Electronics

By

Chirutkar Harshadkumar Govindrao

Registration No. 3535. Date. 31st July, 2006

Research Supervisor

Dr. H. N. Pandya

Department of Electronics,

Saurashtra University,

Rajkot – 360005

May – 2011

In the Feet of

My Sat-Gurudev

Sant Shri Asharamji Maharaj

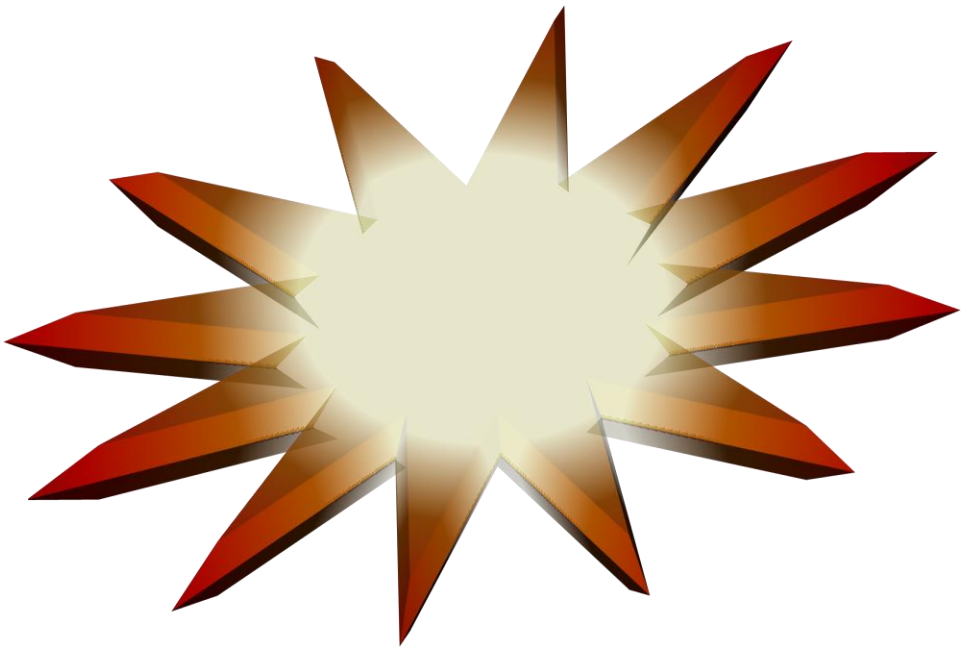
And

His Almighty

Lord of Uncountable Universes

Shri Narayan Hari

ॐ



Statement Under O. Ph.D of Saurashtra University

The content of this thesis is my own work carried out under the supervision of Dr. H. N. Pandya and leads to some contribution in Electronics supported by necessary references.

(Chirutkar Harshadkumar G.)

This is to certify that the present work submitted by Mr Chirutkar Harshadkumar Govindrao for the Ph.D. degree of Saurashtra University, Rajkot has been the result of about five years work under my supervision and is a valuable contribution in the field of Electronics.

Dr. H. N. Pandya
Prof. & Head
Department of Electronics
Saurashtra University,
Rajkot-360005,
Gujarat-INDIA.

Acknowledgement

Here comes a day that I have fulfilled my long cherished 10 years' dream. During these many couple of years of research in Department of Electronics, Saurashtra University, Rajkot, great men supported me to complete my research.

Dr. H. N. Pandya, as a head, guide and as human being understood my nature of curiosity to work and allowed me to work under him. He also helped me to get Rajiv Gandhi Research fellowship from UGC. I had a good support from his family Mrs. Kiran and Ms. Sweta. It is due to only his support I could enjoy my research period and he made my term a golden period of my life. In simple words he fulfilled my dream in all respect and words of thanks are small.

Starting my research period with Dr. A.A. Bhaskar I had a good start to my work and stay at department and Girnar Hostel. Dr. Bimal Vyas, Dr. Maulin Nanavati, Dr Darshan Vyas and Dr. Kapil Bhatt provided all their guidance and experience that was very valuable during the research period.

My sincere thanks to Mr. Hiteshbhai, Mr. Manishbhai, Mr. Dhirubhai, Mr. Pravinbhai, Mr. Vamanbhai, Mr. Tareshbhai, Mr. Jayubhai, Mr. Sandeep, Dr. Pranav, Dr. Anchal, Dr. Khut, Dr. Nikesh sir, Dr. Mahesh sir, Dr. Thumar sir for helping me at the department and university. I also extend my thanks to Mr. Mahesh, Ms. Seema, Mr. Kamal in the laboratory work.

My father Mr. Govindrao and mother Miss. Manjulaben supported me to stay and work in Rajkot. I am very much thankful to my wife Miss. Bhumika, my son Mr. Aavartan and my sisters Mrs. Jayshree and Ms. Poonam as they controlled their temper when I was busy working with my laptop and kits and unable to attend them.

Contents

Chapter 1

Introduction

Chapter 1 Introduction

1.1 Background

The history of computers can be divided into four periods: the mainframe, the mini, the microprocessor and the post-microprocessor. The mainframe era was characterized by computers that required large buildings and teams of technicians and operators to keep them going. Perhaps both academics and students had little direct contact with the mainframe. During the mainframe era, academics concentrated on languages and compliers, algorithms and operating systems. The minicomputer era put computers in the hands of students and academics, because university departments could now buy their own minis. As minicomputers were not as complex as mainframes and because students could get direct hands-on experience, many departments of computer science and electronics engineering taught students how to program in the native language of the computer – assembly language. In mid 1970s, assembly language programming was used to teach both the control of I/O devices and the writing of small programs. The explosion of computer software had not taken place, and if you wanted software you had to write it yourself.

The microprocessor was introduced in the late 1970s. For the first time, each student was able to access a real computer. Unfortunately, microprocessors appeared before the introduction of low-cost memory (both primary and secondary). Students had to program microprocessors in assembly language because the only storage mechanism was often a ROM with just enough capacity to hold a simple single-pass assembler. The advent of the low-cost microprocessor system (single board) ensured that virtually every student took a course on assembly language. Even today, most courses in computer science include a module on computer architecture and organization and teaching students to write programs in assembly language forces them to understand the computer architecture. The 1990s is the post-microprocessor era. Today's personal computers have more power and storage capacity than many of yesterday's mainframes and they have a range of powerful software tools that were undreamed of in the 1970s. The availability of high-

performance hardware and the drive to include more and more new material in the curriculum has put pressure on academics to justify what they teach. In particular, many are questioning the need for courses on assembly language. The post-microprocessor era can also be called microcontroller era.

This leads to the design and development of educational trainer systems. Since the introduction of Intel 8085A microprocessor, there has been explosion of educational trainer systems, evaluation boards and single board computers (SBC). Many vendors started designing microcontrollers on various architectures. The vendors initially concentrated on 8 – bit chips and later on gradually 32 – bit chips. The vendors also started giving technical support with evaluation boards and software for faster development and utilization of their microcontrollers in the market. The customers needed to develop product in time before any other could do.

So, it was mandatory to use such trainer systems or evaluation boards with software support by the design engineers. The educational institutions started inculcating microprocessor and microcontroller in their curriculum because the industry started absorbing basically trained personnel in them. These institutions also demanded trainer systems that are easy to work with and they could complete the syllabus in one or two semesters with practical hands-on training on them. The purchase of trainer systems is costly affair for some institutions and some enthusiast students who desired to excel in microcontroller programming in the student life.

A typical microcontroller trainer system includes evaluation board with extra peripherals and microcontroller on it, programmer hardware, a text editor, cross-assembler/complier, programming software and debugger. The Integrated Development Environment (IDE) software is single window application of PC through which a program can be written, complied/assembled, programmed into the microcontroller and debugging can be done on the written program

1.2 Rationale

To extend the interest of design engineers and research students a PCI bus based microcontroller trainer system is designed, developed and tested. The following are the main facts this type of systems is developed:

- Use of trainer without any hassle of power supply as the trainer system is a PCI card.
- Use of trainer without any hassle of programmer and debugger and their power supplies as they are embedded on the PCI card.
- The needed external peripherals are embedded on the PCI card, so no extra connectivity is need.
- The PORT pins are provided for external use and prototype area with power supplies is also provided.
- This PCI card is supported by the IDE and the device drivers to install this card.

The functional features of the PCI card used as microcontroller trainer system are:

- The 8 – bit AVR microcontroller from Atmel is used as target microcontroller in the PCI card.
- The AVR Studio IDE from Atmel is used as development environment.
- The open source gcc compiler for AVR is used to develop programs in C.
- The JTAG programmer and debugger is incorporated within the PCI card and is also supported by the IDE.

The figure 1 shows typical functional block diagram of PCI card useful for microcontroller trainer.

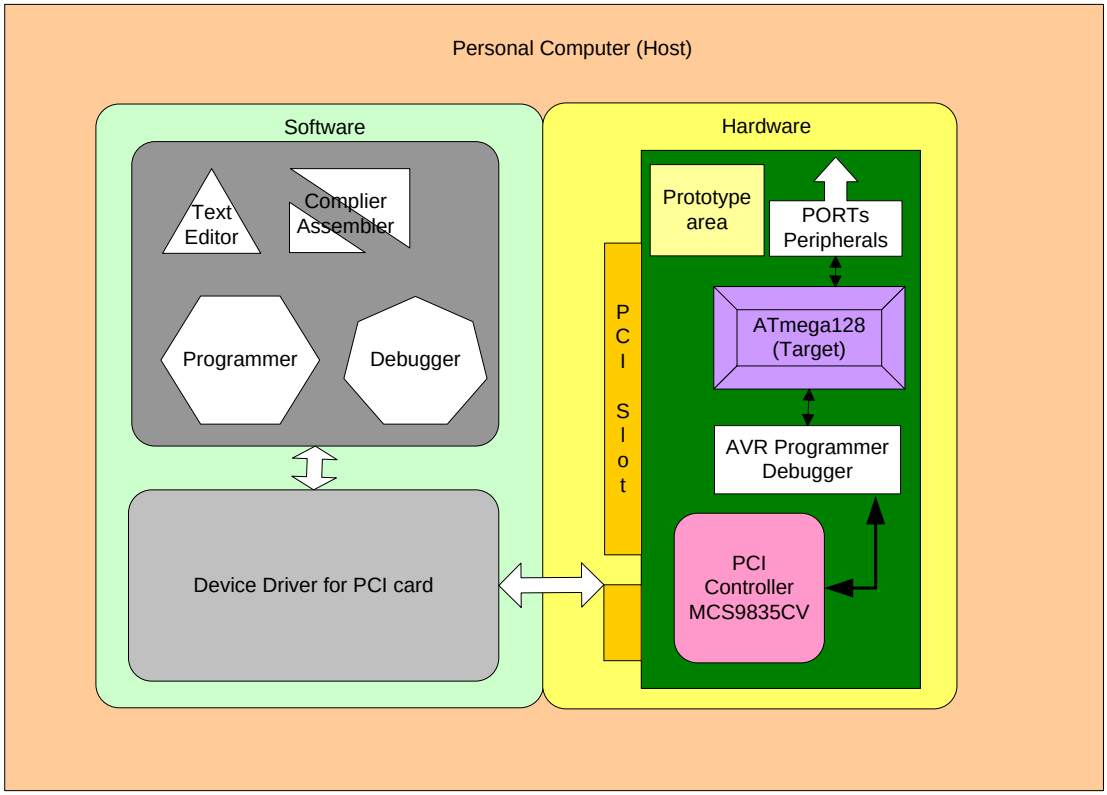


Figure 1.1 Typical Block diagram

1.3 Structure of Thesis

The thesis has been organized into different chapters for the ease of reading and understanding. Care has been taken to follow a particular pattern in such a way that things which are relevant at the beginning and are required to understand the later chapters have been discussed in the first few chapters. A reader having very little knowledge about the subject can easily follow the chapters and understand the details in the subsequent chapters. The thesis for the said title is documented in 10 chapters and each chapter is in its worth to elaborate the development process of the PCI card.

The chapter 1 will have background for the research work as well as the motivation factor. The thesis starts with a general idea to develop PCI card based microcontroller trainer system, instead of an ordinary trainer systems that is available in the market. The student community today mostly faces the problem of learning microcontrollers in short term of a single semester. Learning architecture of microcontroller, software models and hardware models, programming in assembly or higher languages in a semester brings lot of pressure on them to build hardware and software for any project. So, I have tried to present here a complete system that includes programming environment i.e. Integrated Development Environment (IDE), programmer, debugger and some space onboard to build customized application. This chapter also discusses the outline content of the thesis.

The chapter 2 will have description about the objectives and planning done for the work. It will also include learning outcomes, methodology, feasibility of the research work and important tasks to be completed first. How the trainer will be useful to the students will be described here. The development methodology and feasibility of the PCI bus related designs and selection of the microcontrollers will also be discussed. In the beginning of research work the objectives to be accomplished from the work and the learning outcome to be mentioned is important and this is discussed in this chapter. Before proceeding towards the concrete work, the feasibility analysis is to be performed in the sense of technology, cost, cost to time consumption, limitation of hardware/software resources etc. A methodology is required for

this type of work in order to structure the research, design, implementation, and testing that will be carried out. A large number of methodologies exist, although most of them suited to development teams rather than individual projects. For this reason, agile development methodologies are likely to be the most suitable, due to their 'lightweight' nature.

The chapter 3 will describe the efforts done in the literature survey segments. It contains about the existing development tools, PCI bus based cards, Journal papers reviews, Current bus system and related work, programmer and debuggers available in the market, Integrated Development Environment(IDE) for developing microcontroller based systems, programming protocols for programmers and debuggers and at last the outcome of literature survey. Full details about the PCI bus and add-on cards being used today to learning purpose is discussed. Important points noted during the study of different PCI cards, various microcontroller programmers and debuggers and related work being done will be described. Although literature survey is the introductory process before starting research work, but it does not completely end until the targeted aim is achieved.

The chapter 4 is introduced to have a firm theoretical background based on the literature survey done. Technical details of SPI and JTAG programming protocols for programmers and debuggers are given here. The architecture and the internal peripheral of AVR 8 – bit microcontroller from Atmel will be described here. The programming peripherals of AVR will be discussed in connection to our PCI chip. The PCI interface chip and its working and its design will be discussed. The background theory of PCI bus is also discussed here.

The chapter 5 will discuss the Requirement Specification and Analysis of the research work, about the IDE, PCI Chip and its drivers, Interfacing hardware, selection of microcontroller and finally the testing the modules to be developed. The research work done in the PCI chips and its driver's area, development to extend the IDE and the reason for selection of microcontroller and which new modules to be developed will be described here in detail. More

over the requirement specifications for all stages of research work will be described in this chapter. The result of this chapter will lead to the procurement of the materials and chips used for this research work.

The chapter 6 will contain the system design and test plan which includes the architectural design, details design and test plan of the detail design. The principle block diagram of the PCI card to be developed will be described here. The main blocks will disintegrate in small blocks that will be decided and then will proceed for the development phase. At what stages the blocks will be tested and how will be tested will be decided. This chapter just shows the flash picture of the development process and will clear picture of the blocks to be worked with.

The chapter 7 will comprise of systems development stages of individual modules i.e. IDE and drivers, PCB of JTAG programmer and debugger and Prototype design of ATmega128 board and its interfacing modules. In short this chapter has its practical importance as this includes the design stages of all blocks described in previous chapter. Programming of the microcontrollers and its PCB design will be taken here in detail. The PCI chip and its driver implementation and its working with the programmer will be described. The target microcontroller ATmega128 and its signal lines will be designed here with its other supporting chips. If required some interfacing modules may also be developed and tested here if needed.

The chapter 8 will contain implementation, testing and deployment of the research work. It will include testing of the PCI controller subsystem, JTAG-ICE subsystem, modules testing, system integration and testing and discussion on testing strategy. In this chapter how each block is tested for working will be described, so as to confirm that there is no fault on next related block. The testing procedure of JTAG systems is also described here. The working of PCI chip in serial modes is shown here. And how do the JTAG works for debugging the target AVR is also discussed here.

The chapter 9 will contains the installation and maintenance of PCI card. It will describe the prerequisite for the PCI card, software. The operating manual that is to be read before installing the PCI card is described here. As the PCI cards are very tricky to be used and installed the operating manual is very useful for a newbie user. The manual includes proper steps to install the PCI card in various operating systems. It also describes how to program the firmware on the programmer chip and diagnose the serial port working of PCI card.

The final chapter 10 will discuss about the evaluation of the PCI card developed, results developed from its use and conclusion. This chapter includes the problems faced regarding various phases of project development, selection of microcontroller, development methodology, language of programming and recommendation for future work.

The appendices will contain Gantt chart, schematics of circuits, bill of materials, datasheets, figures, abbreviations, references used during research work and paper published and presented based on the research work done. The appendix also includes an important sample exercise section for the students learning embedded systems and AVR microcontroller.

The converging computing and communications industries are on the threshold of a parallel-to-serial interconnect transition. This trend, observed in many application areas, has been gaining momentum since the introduction of the PCI Architecture.

The stringent electrical, functional, and timing specifications for PCI cards pose a difficult challenge for designers. Hobbyists and even designers could go in for readymade prototyping boards/cards offered by a host of manufacturers. For those having access to FPGA programmers, Xilinx (in partnership with Compuware Numega, Comit Systems Inc., and Virtual Computer Corp.) offers economical PCI compliant design solutions, which one can use as a starting point for one’s own PCI based project.

PCI is being used in the industry for several years now replacing ISA slots. The PCI Local Bus[2] is a high performance bus with multiplexed address and

data lines. The bus is intended for use as an interconnect mechanism between highly integrated peripheral controller components, peripheral add-in boards, and processor/memory systems.PCI Local Bus offers features and benefits in many areas to achieve multiple price-performance points and can enable functions that allow differentiation at the system and component level. Some of the benefits are,

Robust High Performance Bus – Low latency burst transfer[3] on a Synchronous bus with Hidden central arbitration[3]. Provides parity on both data and address, and allows implementation of robust client platforms;

Low Cost – Optimized for direct silicon with minimal glue logic and in typical ASIC processes, Inexpensive packaging and small number of pins, Small form factor add-in boards;

Platform Independent Plug-and-Play Environment - – Processor independent auto configuration support of PCI Local Bus add-in boards and components. PCI devices contain registers with the device information required for configuration. Full multi-master capability allowing any PCI master peer-to-peer access to any PCI master/target.

In order to enjoy these benefits a large number of adapter cards tend to use single component solutions with integrated PCI interface using ASIC or FPGA technologies. Some other adapter cards use off-the-shelf PCI controller chips with the add-in function. However currently these off the shelf controller chips are targeted to a broad market segment at varying performance levels. Therefore the Add-on interfaces of these controller chips are complex in nature. However, there are still a large number of off-the-self ISA components used in the adapter card market. These ISA chips operate on a very simple ISA-like interface and demand a low cost PCI controller chip with a simple ISA-like interface for building adapter cards. Similarly, prototyping, and small volume designs that may not demand high performance can employ ISA-like interface also with a low cost PCI controller chip having a simple ISA-like interface with the same ease.

The processor system bus continues to scale in both frequency and voltage at a rate that will continue for the foreseeable future. Memory bandwidths have increased to keep pace with the processor. Indeed, as shown in Figure 1, the

chipset is typically partitioned as a memory hub and an I/O hub since the memory bus often changes with each processor generation. One of the major functions of the chipset is to isolate these ever-changing buses from the stable I/O bus.

Close investigation of the 1990's PCI signaling technology reveals a multi-drop, parallel bus implementation that is close to its practical limits of performance: it cannot be easily scaled up in frequency or down in voltage; its synchronously clocked data transfer is signal skew limited and the signal routing rules are at the limit for cost-effective FR4 PCB technology. All approaches to pushing these limits to create a higher bandwidth, general purpose I/O bus result in large cost increases for little performance gain. To PCI's credit it has been used in applications not envisaged by the original specification writers and variants and extensions of PCI can be found in desktop, mobile, server and embedded communications market segments.

Today's software applications are more demanding of the platform hardware, particularly the I/O subsystems. Streaming data from various video and audio sources are now commonplace on the desktop and mobile machines and there is no baseline support for this time–dependent data within the PCI 2.2 or PCI-X specifications. Applications such as video-on-demand and audio re-distribution are putting real-time constraints on servers too. Many communications applications and embedded-PC control systems also process data in real-time. Today's platforms, an example desktop PC, must also deal with multiple concurrent transfers at ever-increasing data rates. It is no longer acceptable to treat all data as equal – it is more important, for example, to process streaming data first since late real-time data is as useless as no data. Data needs to be "tagged" so that an I/O system can prioritize its flow throughout the platform.

Applications, such as Gigabit Ethernet® and InfiniBand®, require higher bandwidth I/O. A third generation I/O bus must include additional features alongside increased bandwidth.

Chapter 2

Objective and Planning

Chapter 2 Objective and Planning

2.1 Objectives.

The primary motivation for this work was the lack of commercially available low cost microcontroller boards that would have all the communication modules or components interfaced on it. Having a single board simplifies the design and reduces the need of using several different boards and components for performing research and experiments. The board is being further used as a platform for teaching and learning embedded systems. The target user groups for the PCI card are the students and researchers in various universities and research labs. The other design objectives were robustness, reliability and functionality of the board. Most of the early technological learners such as high school students lack experience and expert knowledge for interfacing a controller board with other components. To prevent the learners from making errors, connectors on our board have been made foolproof (the user cannot damage the components of the board by plugging cables in the wrong sockets). After reviewing the commercially available micro-controller boards with respect to their suitability as teaching tools, we concluded that none of the existing microcontroller boards met our requirements as per out knowledge. We then designed a new controller board based on previous boards and added various communication modules to it. The major uses of this embedded systems communication board can be summarized as follows:

- The board will be used as a teaching tool in embedded systems courses.
- The embedded systems course will cover the basic microcontroller programming and hardware details related to the board.
- Laboratory exercises form an essential part of any embedded system course.
- This board will serve as basic learning module for the students performing programming, testing and other interfacing exercises in the lab.
- Research in the field of PCI bus and PCI bus interfacing can be performed on the basic level using this board. Measurement of the

throughputs and calculating the maximum throughputs at different speeds and in different conditions can be an excellent research issue to work upon.

- Interfacing of different sensors on the board will allow the user to accumulate data from the sensor and store it through PC.
- The board will provide enough resources for the researchers and developers to build applications on top of it by interfacing different technologies with it.

2.2 Learning Outcomes.

This research work provides a chance to gain in-depth knowledge of the protocols used in the end application, such as JTAG, SPI, TWI, UART protocol. Also some digital design skills can be learnt through implementing interfaces such as PCI (Peripheral Component Interconnect) & JTAG from an embedded microcontroller. Implementing the software system will require a large amount of code to be designed, written and tested across two different hardware platforms (embedded hardware & PC). It also involves production of a system that will likely include many surface mount and small pitch components, thus providing the change to increase PCB layout skills.

2.3 Feasibility

The feasibility analysis is very important task before taking up any research work or project. In terms of feasibility means the work taken up requires some entities and are we capable to fulfilling the needs and how we are capable. The feasibility analysis has been done on the following issues:

- **Economical feasibility**: The electronic components, PCB fabrication Laboratory, Advance microcontroller/microprocessor laboratory, evaluation kits and software, PC were provided by Department of Electronics, Saurashtra University, Rajkot. So, there were no hindrances in purchasing of required material.
- **Technical feasibility**: Some evaluation boards that we wanted to purchase, the technical support from the company was not provided. Then 1 ½ years later we got a PCI chip MCS9835CV[5] that was an

Indian make chip from MosChip Semiconductor Technology[6]. We also got full technical support for the chip and they also provided us two sample chips for research and development. So, finally we decided to make it for MCS9835CV from MosChip Semiconductor Technology.

2.4 Methodology.

Before you start a System development project, it is important to decide on a suitable development strategy. This topic discusses the possible development strategies for the implementation of the Development of PCI Embedded Card useful for Microcontroller Trainer. For this discussion we assume that the development team itself is already set up. This means that the number of team members, their skills, their responsibilities, and so on are already defined. We start by discussing why there is a need for a System development model, and then we review the most commonly used System development models. After that, we'll choose the model that best suits the said project. Our project can be called a hardware-software development project.

It is estimated that one-third of all hardware-software development projects are cancelled before the system is implemented. Of the remainder, two-thirds significantly overrun their budgets. Research also shows that more than 80 percent of all project errors are committed in the critical analysis and design phase before actual phases are designed. Even though these numbers may vary (depending on who publishes them), they show clearly that there is an essential problem with system development. What is the root cause of the problem? How can hardware-software development be made more predictable? [7]

Looking at other engineering disciplines, such as civil engineering, we can see that a structured approach seems to be one of the keys for predictability and repeatability and therefore success. Let's take the example of building a house. To build a house, people typically follow a structured approach. First the budget is secured and then the land is acquired, followed by application for the necessary permits before a detailed plan of the house is developed. All these tasks are necessary to build a house within the constraints of time, cost, space required, and options chosen. It is very unlikely that someone would ask a contractor to build a house without agreeing on a plan that considers all

the constraints. This approach seems to make sense. It has worked and will work for building many new houses.

Analyzing this kind of approach reveals some of the deficiencies in hardware-software development. Often, software developers write a program or code without having any plan in the form of a functional or design description (not to mention a refined, detailed plan). When that happens, the code often does not reflect constraints such as budget, functionality, modularity, reusability, maintainability, and available technologies. It is as if a building contractor were starting to build a house without knowing the location of the house and the architectural plan.

It is at this point that software engineering methodologies come into play. Software engineering methodologies try to structure software development in the same way other engineering practices are structured to make software development more predictable and repeatable, and therefore more successful. We can build our project using software engineering methodologies as we develop software for our system along with hardware.

Some commonly used software engineering methodologies and models: Code and Fix, Waterfall, V, Spiral, Staged Delivery, and Evolutionary Prototyping. Then we look at the more dynamic models: Scrum, Adaptive Software Development, the Unified Process (UP), and Extreme Programming[7].

A software development model should not be dumped blindly onto the project but rather should be carefully chosen according to the project, the environment, and your goals. After a certain model is chosen, it must be adapted to fit the project and organizational culture. All the models mentioned here are flexible enough that you can adjust them to the environment they are used in. This Waterfall Software Development Model suits our research work.

The Waterfall Software Development Model

Characteristics

The Waterfall model is considered the grandmother of all the software engineering models and is the most well known of all software engineering methodologies. The Waterfall model works well in an environment of well-understood requirements. For the model to work well, you need to make sure that you eliminate the possibility of a lot of midstream requirements changes.

This avoids a common source of problems when using this development model.

Definition

The Waterfall model is a sequential, document-driven methodology. To advance from the current phase to the next phase, the project team must review and release an artifact in the form of a document. The release of the corresponding document triggers the end of one phase and the beginning of the next (see Figure 2.1).

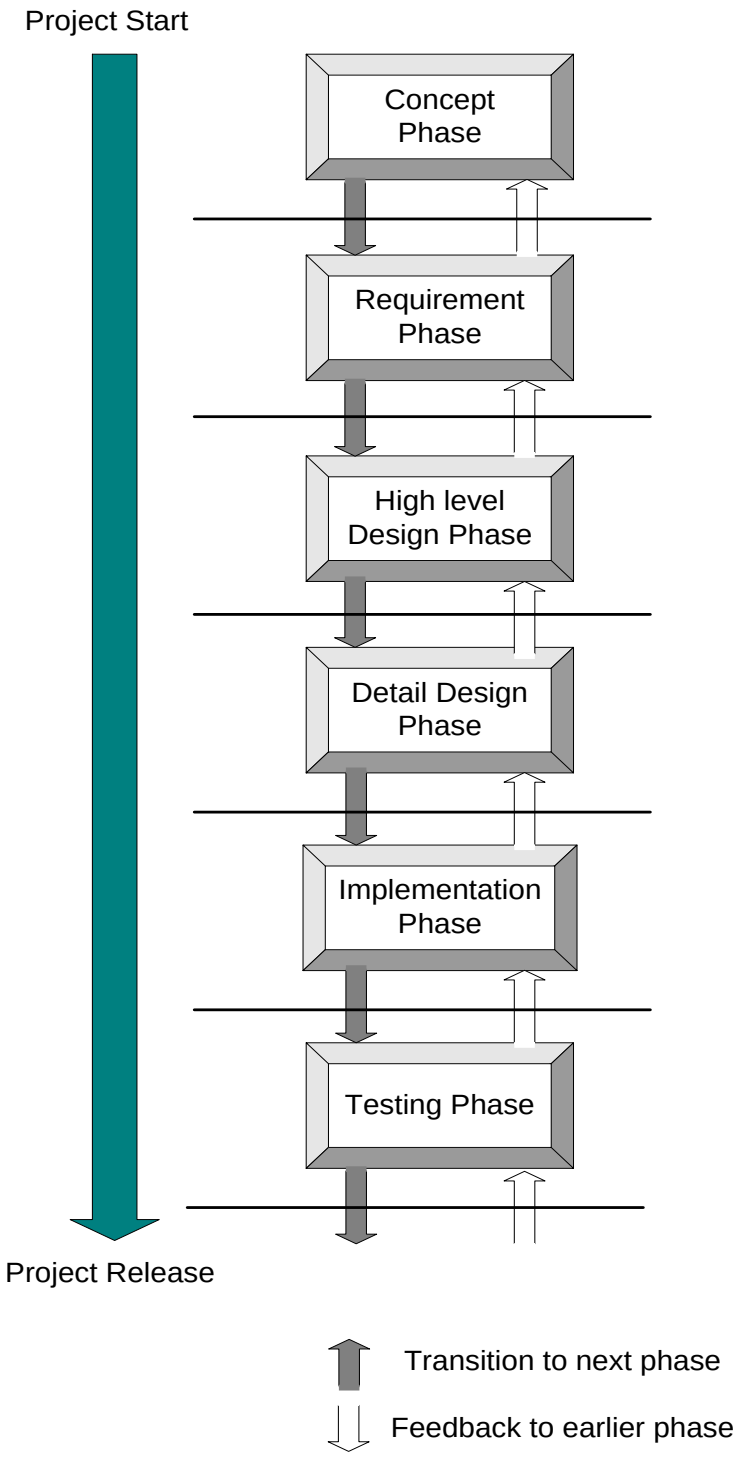


Figure 2.1 Waterfall Software development model

The Waterfall software development model uses sequential, non-overlapping phases and carries over documents from phase to phase. The project flow is as follows:

1. **Concept phase:** In this first step the project goals and constraints of the project are discussed, and the initial effort estimate is provided. The phase ends with the release of the concept document.
2. **Requirements phase:** After the concept document is reviewed and agreed upon, the requirements are engineered. The project team must completely understand the system, technologies, and constraints at this point in order to define a complete set of requirements for the system. The transition to the next phase occurs after the requirements document is approved.
3. **High-level design phase:** In this phase the system's overall architectural design is developed, and the functional modules are identified. At the end of this phase, the high-level design document is the milestone necessary to proceed to the detailed design phase.
4. **Detailed design phase:** After the functional modules are identified, the detailed design of these modules is worked out. At the end of this phase the detailed design document is released, providing the base for the implementation by the developers in the next phase.
5. **Code implementation phase:** The system is implemented according to the system documents produced in earlier stages. In many projects, unit testing is also part of this phase. At the end of the phase the code is implemented and is unit-tested as proof of the (correct) implementation.
6. **Test phase:** After implementation and unit testing of the code, the integration and acceptance testing takes place to verify that the system works as described and agreed upon. The test documentation and the passed test records are the final documents that are released before the product is shipped to the customer.

In the pure Waterfall model, the document of a given phase must be completed, reviewed, and released before the project can proceed to the next phase. It is easy to see that the process assumes that the problem, the domain, and the technologies are well understood. You must have this understanding in order to complete the planning before the implementation can start. If, during later phases, a problem is found that requires changes to

previous phases, the project must roll back and correct all the affected documents before it can proceed.

Pros

The Waterfall model works well in projects that have stable requirements in a known technical domain, even with complex tasks. The Waterfall model enables planning and analysis of complex problems early in the project and therefore removes some of the risk from later phases, when it is much more expensive to fix problems. The model also works well with inexperienced or technically weak project teams because it adds a structured, well-planned approach to the project. Therefore, it minimizes unnecessary work due to inexperience or lack of technical knowledge.

Cons

On the downside the Waterfall model is sensitive to changes in phases that are already finished. It is often difficult to specify all the requirements before you do some of the design work. If it's necessary to make changes to requirements after the initial design work, the project must revisit the previous phase and adjust the work (or documentation) of that phase. These changes might trigger changes in the work that has been done in the earlier phase and consequently to its work artifact. Before the project can continue to the next phase, all the changes to all previous phases continue to be propagated upstream until they have been incorporated. It is easy to see that changes made midstream can cause massive rework.

Variations

The Sashimi model is named after the Japanese style of presenting fish in overlapping slices. This approach modifies the Waterfall by allowing for overlapping phases. The main advantage is that it reduces documentation and gives you the freedom to proceed with incomplete phases. The problem with this model is that the milestones are no longer obvious, and progress cannot be tracked as easily as with the pure Waterfall model.

Another variation, the Waterfall with Subprojects approach, allows for subprojects that can be executed independently. This lets you implement easily implemented features at an earlier stage, before the more complex and unknown features are completely planned. This approach is problematic if unforeseen dependencies are uncovered during later stages.

Chapter 3

Review of Literature and tools

Chapter 3 Review of Literature and tools.

This chapter provides the review of literature survey, evaluation tools and technologies existing in the market at the time of requirement phase of this research work.

3.1 Existing Development Tools.

To develop any product or evaluation system using microcontroller/microprocessor and supporting peripheral chips, technical support is mandatory requirement. The technical support includes:

- The detail datasheet or family product book.
- Special electronic designing criteria.
- PCB designing tips.
- Evaluation schematic.
- Application notes for using all of its peripherals including sample firmware programmer and debugger.
- Integrated development environment.

Moreover, some vendors have started online support and forums so that we can post our questions to them and get the answer by their technical experts. We can also join a forum or start forum for more detail discussion of our problems.

A budding electronics student needs practical learning through education trainer system to solve the real-world problems. Now the colleges have started realizing the potential of hands-on-training. Most of the MNCs are promoting their products in universities through university programs, while enabling finishing schools to cater to the industry need by providing industry-ready trained candidates as part of their business strategy.

India has the third largest higher-education system in the world, next to China and the US. The government spending on education is close to 4.5 % of the GDP. It is estimated that during 2010-15, more than 90 million youths will enter the working-age population. With precisely tuned products, educational and hobby kits market can grab a large share of the technical education sector.

Some of few easily available microcontroller trainer systems are discussed now.

Frontline electronics, Salem, Tamil Nadu has about 15 years of experience using 8051 microcontrollers and they provide Topview Simulator, Topview Programmer, Topview debugger and many low cost evaluation boards for 8051 and ARM family microcontrollers. The figure 3.1 shows an example of Topview programmer.

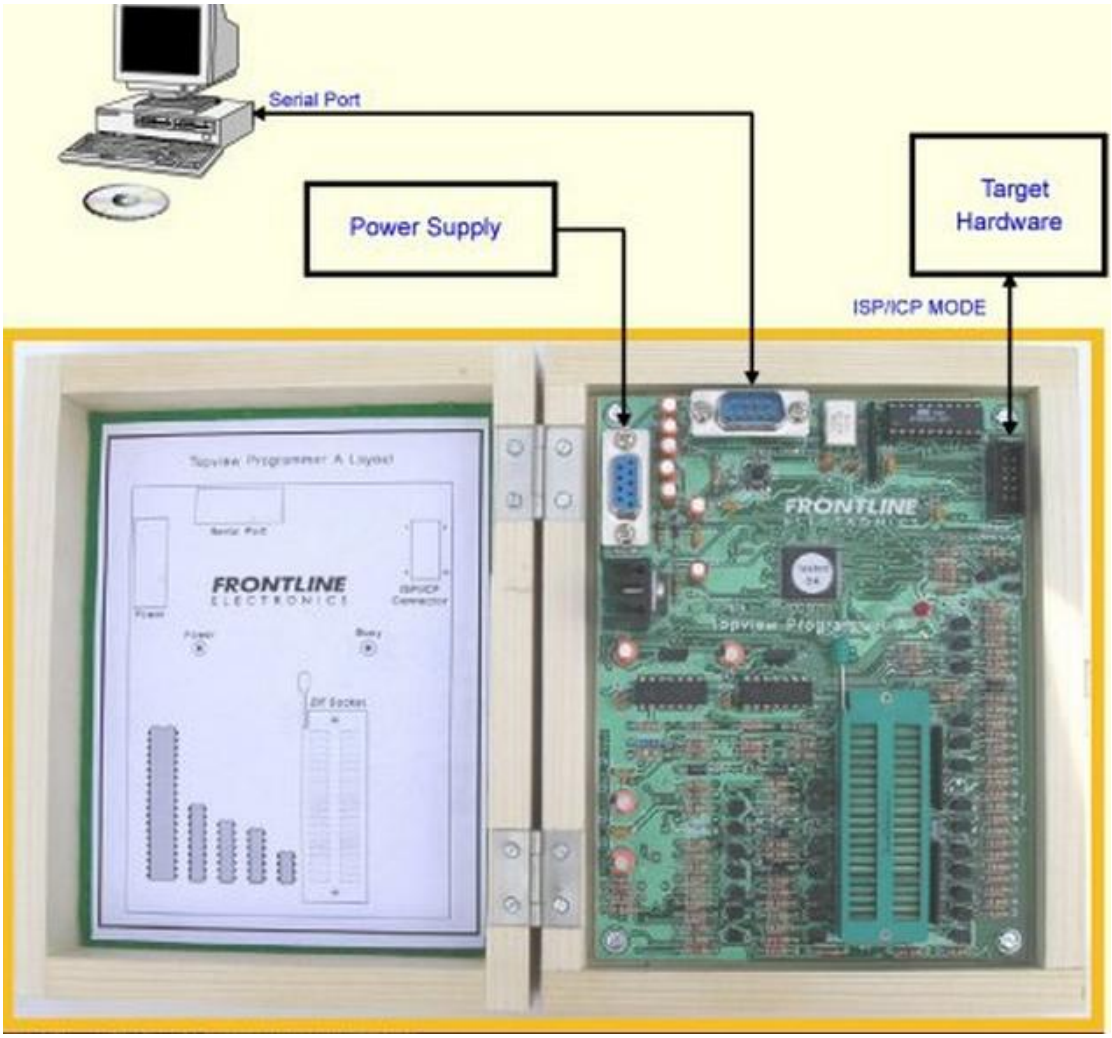


Figure 3.1 TopView Device Programmer

ESA systems, Banglore also provides educational trainer systems, programmers and Keil products. The figure 3.2 shows an example of education trainer system provided by ESA.

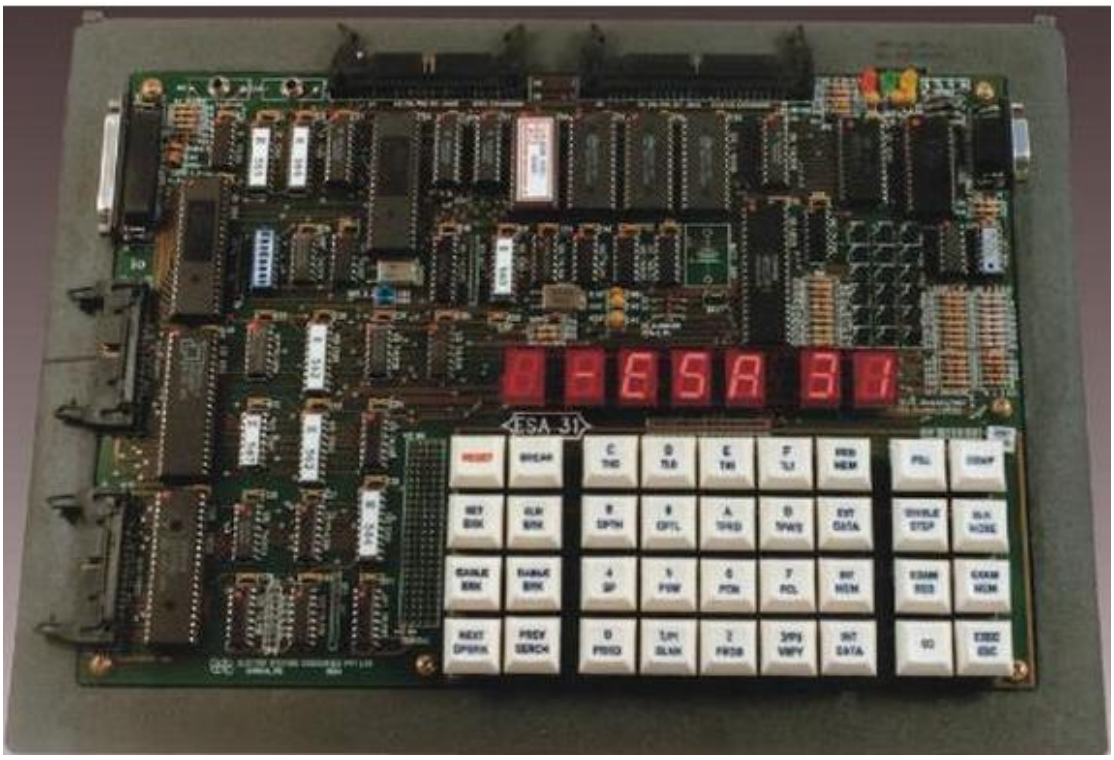


Figure 3.2 ESA31 system

The microcontroller trainer systems can be classified into three categories based on the requirement and learning topology of the user.

1. Simple trainer systems with serial/parallel port for programming and interfacing to PC.
2. FPGA based trainer system with advance peripherals on board with USB and Ethernet interface that may include ZigBee wireless chipset.
3. PCI Bus based trainer systems that may include configurable FPGA/CPLD.

For this research work, we have selected to develop a PCI Bus based trainer system.

PCI cards are usually implemented by one of three approaches:

1. Using an ASIC which includes a PCI interface core as well as other functionality.

This approach is used for high volume applications where cost is the driving factor, and where a large gate count is needed to implement the design. In these applications, the PCI interface core is usually small compared with the custom logic.

2. Using a standard PCI interface chip.

This approach is less cost effective than using an ASIC (for high volume applications), but it is practical when there is a need to interface existing chips to the PCI bus (for example, a DSP board for PCI), with small amounts of custom logic. These chips are available from a few companies with many models, some offering full master/target capabilities, and some offering target only. Due to their general-purpose nature, however, they may not offer the best performance for all applications.

3. Implementing a PCI interface using a CPLD or an FPGA.

This approach is used when prototyping ASIC designs, or when there are special requirements from the PCI interface precluding the use of a standard PCI interface chip, or when low quantities are desired, precluding the use of an ASIC. for example, a design requiring a very low latency response may use a CPLD instead of a standard PCI interface chip.

The survey on these types of PCI cards and chips is included in the next section 3.2

3.2 PCI Bus based Cards.

Almost thirteen PCI Bus based evaluation cards and chips had been surveyed. The detail discussion about these cards /boards will proceed in next paragraphs.

3.2.1 Toshiba **TX System RISC Development Tools**

Product Name: TMPR4925/26XB-200

The main reference board consists of TX4925/26 MIPS RISC Microprocessor, a highly integrated ASSP solution based on the Toshiba TX49/H2 processor core, a 64-bit MIPS I, II, III Instruction Set Architecture (ISA) compatible with additional instructions. The TX4925/26 has integrated peripherals such as SDRAM memory controller, NAND Flash memory controller, PCI controller, AC-Link controller, PIO, SIO, SPI, CHI, PCMCIA I/F and timer. This class of product is targeted for applications that require a high-performance and cost-effective solution such as networking, digital consumer and Internet appliance. The figure 3.3 shows the block diagram of the reference board.

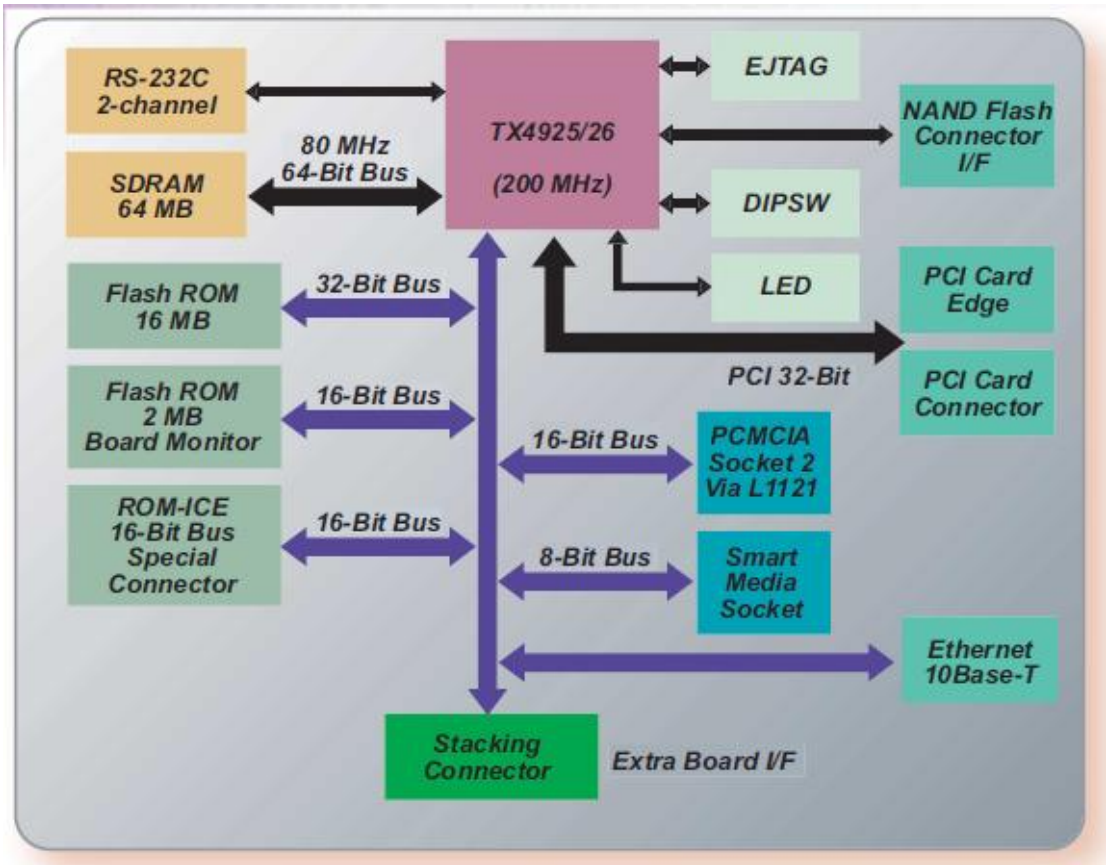


Figure 3.3 TX4925/26XB Reference Board Block Diagram

3.2.2 TC86C001FG (GOKU-S) Based Board

The TC86C001FG (GOKU-S) is a highly integrated and high performance solution. It provides most popular and widely used 5 interfaces – PCI Interface, ATA/ATAPI Host Controller, USB Host Controller, USB Device Controller, I2C bus or Serial I/O Interface. The GOKU-S provides the best solution for a wide range of digital products such as set-top boxes, personal video recorder, information appliances, Multifunction Printer. The figure 3.4 shows the block diagram of TC86C001FG (GOKU-S).

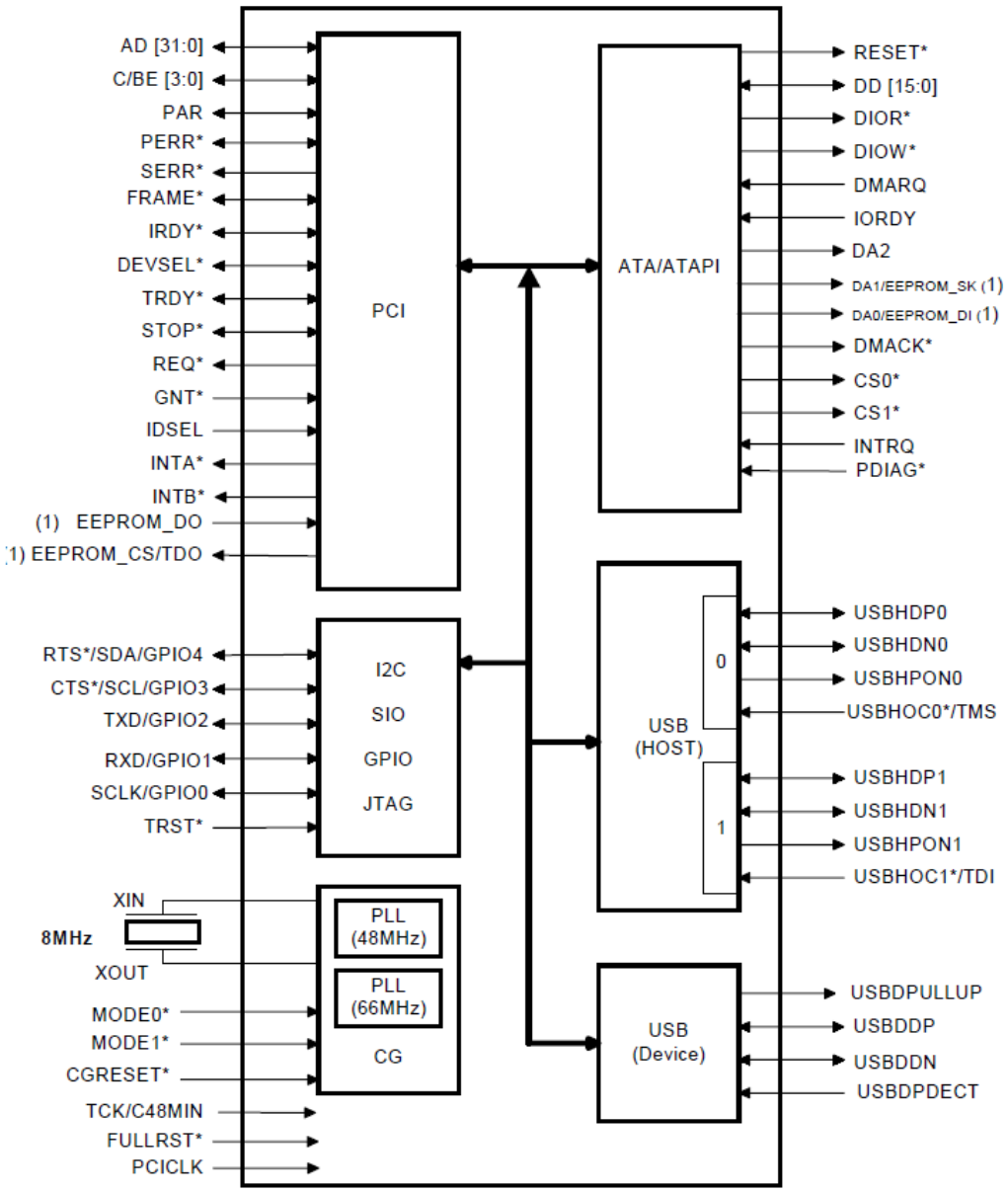


Figure 3.4 Interface diagram for TC86C001FG (GOKU-S)

3.2.3 Actel Peripheral Control Interface Cores CorePCI for FPGAs

Actel PCI cores with nonvolatile FPGA devices offer a flexible and affordable alternative to standard PCI integrated circuit devices. In fact, standalone PCI chips are often more expensive than an Actel PCI core with ProASIC3/E. The cores also enable you to integrate multiple elements onto one device to reduce the system cost as well as the time-to-market. All these can be done without the need for expensive NRE payments required for an ASIC. CorePCIF and CorePCI are the only cores available for FPGAs that offer 33

or 66 MHz, 32- or 64-bit bus width, and target, master, and master/target configurations, all in one product. Actel provides 66/64 PCI in a value-based FPGA, even with industrial temperature operation. The figure 3.5 show the picture of **CorePCI Evaluation Board.**

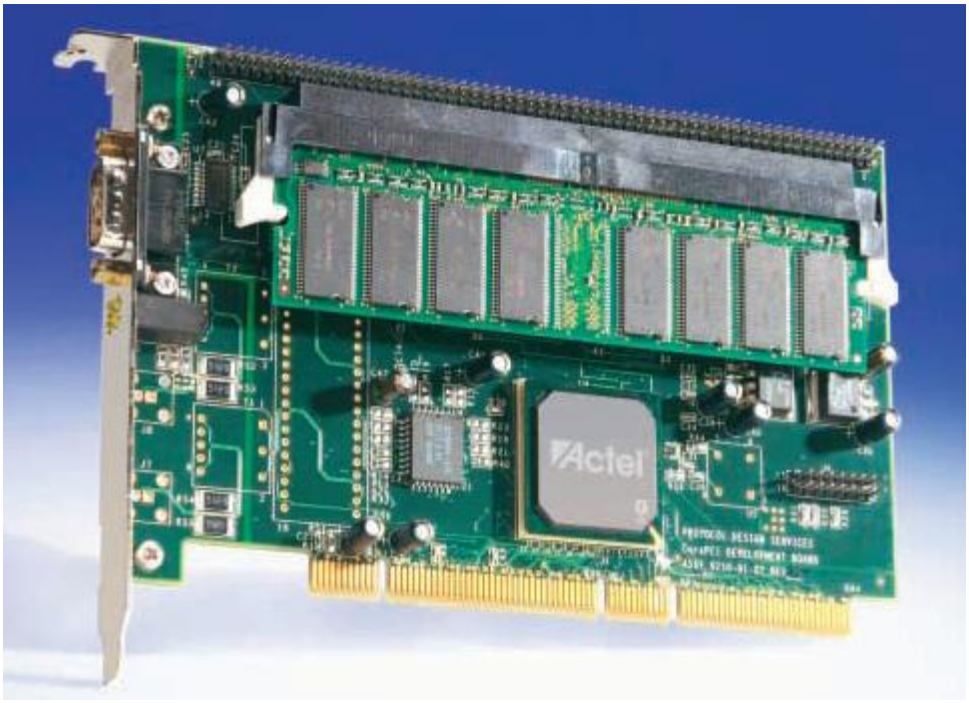


Figure 3.5 Picture of CorePCI Evaluation Board.

3.2.4 CY7C09449PV PCI Bus controller from Cypress semiconductor.

The CY7C09449PV is one of the PCI interface controllers in the Cypress Semiconductor PCI-DP™ family. The CY7C09449PV provides a PCI master/target interface with direct connections to many popular microprocessors. It provides 128 Kb of dual-port SRAM that is used as shared memory between the local microprocessor and the PCI bus. An I_2O message unit complete with message queues and interrupt capability, is also provided. The CY7C09449PV allows the designer to interface an application to the PCI bus in a straightforward, inexpensive way. The figure shows the block diagram of CY7C09449PV PCI Bus controller.

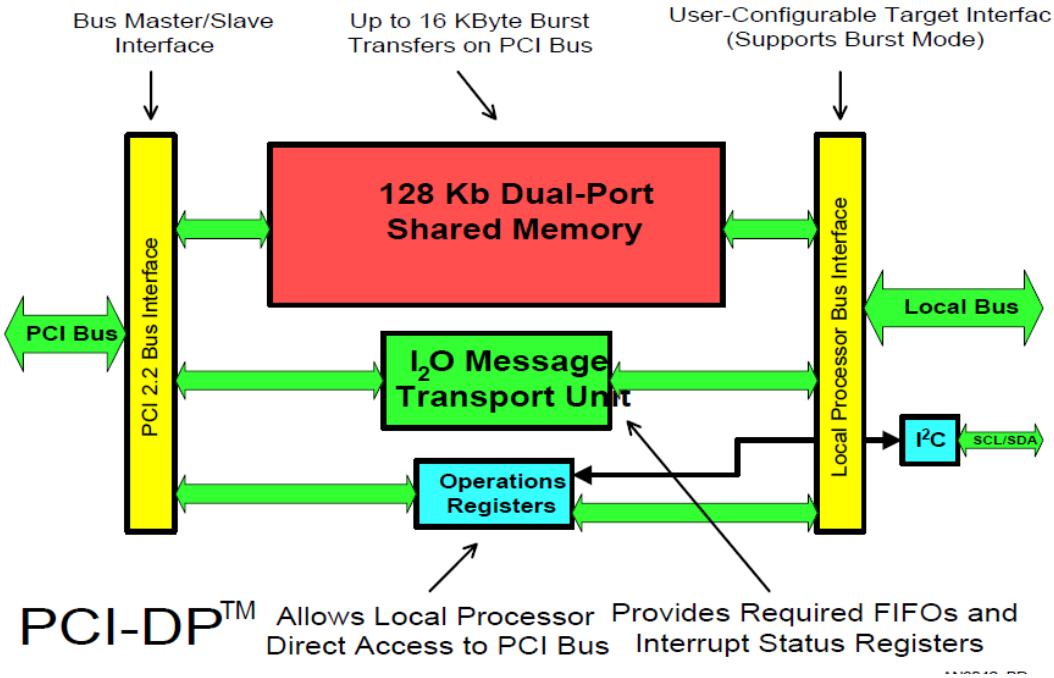


Figure 3.6 Block diagram of CY7C09449PV PCI Bus controller

3.2.5 MCF547x/8x EVB Evaluation Board from Freescale semi.

MCF5470 has High performance (400+ MIPS) V4e ColdFire core with MMU / FPU

Support, 32-bit 133MHz DDR memory controller, configurable as SDRAM Controller, Hardware accelerated encryption (DES, 3DES,AES, MD5, SHA-1, RNG), 32-bit V2.2 PCI interface 33/66MHz, 32KB on-chip SRAM, Two fast Ethernet controllers, 16 channel DMA controller, Seven timers, including watchdog, Four programmable serial controllers, Two FlexCAN controllers (MCF548x), USB 2.0 device with integrated PHY, 1.5V core, 2.5V DDR, 3.3V I/O, Industrial temperature range -40°C to +85°C. The key applications of this processor are Point of Sale terminals, Network attached storage, Security access control systems, Building automation systems, Factory automation systems, Process control equipment,

MCF547x/8x EVB Evaluation Board is the board supporting this microcontroller. The figure 3.7 show block diagram of MCF5470 *32*-Bit ColdFire Microcontroller

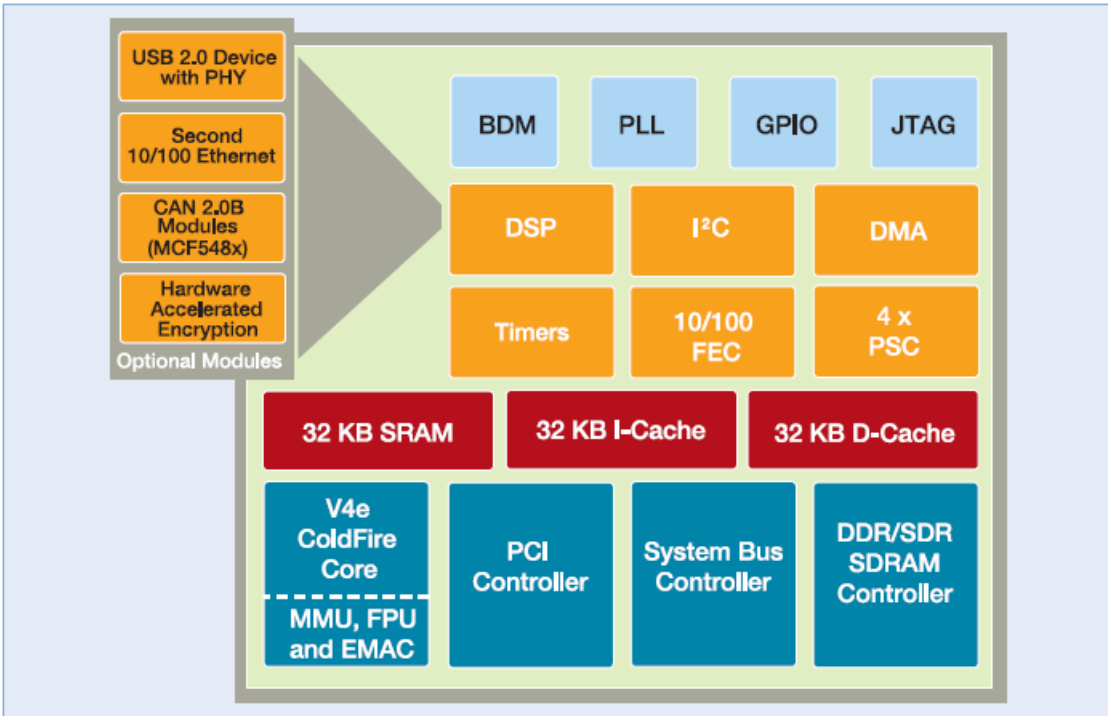


Figure 3.7 block diagram of MCF5470 *32*-Bit ColdFire Microcontroller

3.2.6 QuickWorks Tool Suite from QuickLogic Corporation

The QL5022 device in the QuickLogic® QuickPCI™ ESP (Embedded Standard Product) family provides a complete and customizable PCI interface solution combined with 25,000 system gates of programmable logic. This device eliminates any need for the designer to worry about PCI bus compliance, yet allows for the maximum 32-bit PCI bus bandwidth (132 MBps). The programmable logic portion of the device contains 387 QuickLogic logic cells. The QL5022 device meets PCI 2.2 electrical and timing specifications and has been fully hardware tested. The QL5022 device features 3.3 V operations with multi-volt compatible I/Os. Thus, it can easily operate in 3 V systems and is fully compatible with 3.3 V, 5 V, or Universal PCI card development.

Software support for the QL5022 device is available through the Quick*Works*□ development package. This turnkey PC-based Quick*Works* package, shown in Figure 3.8, provides a complete ESP software solution with design entry, logic synthesis, place and route, and simulation. Quick*Works* includes VHDL, Verilog®, schematic, and mixed-mode entry with fast and efficient logic synthesis provided by the integrated Synplicity Synplify

Lite tool, specially tuned to take advantage of the QL5022 architecture. Quick*Works* also provides functional and timing simulation for guaranteed timing and source-level debugging. The UNIX-based Quick*Tools* and PC-based Quick*Works* provide a solution for designers who use schematic-only design flow third-party tools for design entry, synthesis, or simulation. Quick*Tools* and Quick*Works* read EDIF netlists and provide support for all QuickLogic devices.Quick*Tools* and Quick*Works* also support a wide range of third-party modeling and simulation tools.

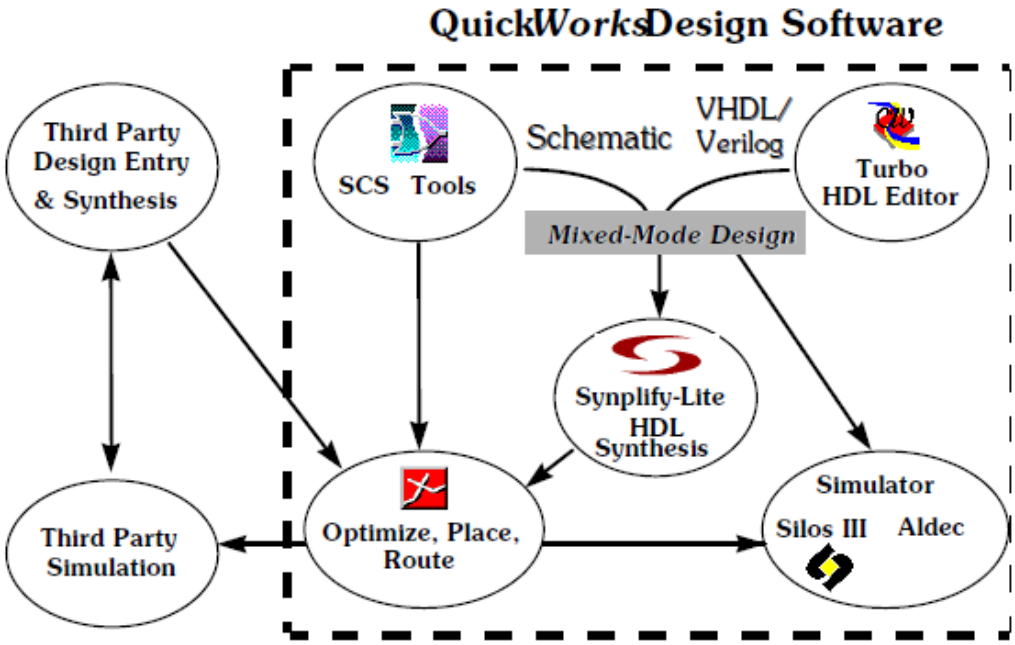


Figure 3.8 *QuickWorks Tool Suite*

3.2.7 Texas Instruments PCI1510

The Texas Instruments PCI1510 device, a 144-terminal or a 209-terminal single-slot CardBus controller designed to meet the *PCI Bus Power Management Interface Specification for PCI to CardBus Bridges*, is an ultralow-power high-performance PCI-to-CardBus controller that supports a single PC card socket compliant with the *PC Card Standard* (rev. 7.2). The controller provides features that make it the best choice for bridging between PCI and PC Cards in both notebook and desktop computers. The *PC Card Standard* retains the 16-bit PC Card specification defined in the *PCI Local Bus Specification* and defines the 32-bit PC Card, CardBus, capable of full 32-bit data transfers at 33 MHz. The controller supports both 16-bit and CardBus PC

Cards, powered at 5 V or 3.3 V, as required. The controller is compliant with the *PCI Local Bus Specification*, and its PCI interface can act as either a PCI master device or a PCI slave device. The PCI bus mastering is initiated during CardBus PC Card bridging transactions. The controller is also compliant with *PCI Bus Power Management Interface Specification* (rev. 1.1).

3.2.8 PCI 9052RDK-LITE (RDK-LITE) from PLX Technology.

The PCI 9052RDK-LITE (RDK-LITE) Rapid development Kit delivers a complete solution for converting existing ISA bus add-in cards to PCI target adapters. The RDK-LITE board features an ISA slot connected to the PCI 9052 ISA bus port. Designers can plug their existing 8-bit or 16-bit ISA cards into this slot to begin their PCI adapter software development on a proven hardware platform. A prototyping area featuring 30 surface-mount footprints provides space to add additional memories, FIFOs, ASICs and I/O devices. This prototyping area allows designers to develop their own custom hardware without having to wait for fabrication of their own PCI-compliant boards. The RDK-LITE kit includes the PLX Hardware Development Kit (HDK) CD, containing reference design information for hardware development. Also included is the PLX Software

Development Kit Lite Edition (SDK-LITE), providing a complete Microsoft Windows host-side development environment. The RDK-LITE hardware reference board serves as both hardware and software development platforms for PCI 9052 based designs. The board ships pre-configured for ISA bus mode and de-multiplexed address and data bus operation (C Mode), but it is user configurable for multiplexed address and data bus operation (J Mode). Its local bus memory controller and SRAM enable immediate Direct Slave and DMA code development and testing. Its large and flexible prototyping area enables the easy extension of the test and debug features of the RDK to include your value added logic.

The HDK CD includes complete documentation of the RDK board design, making its components easily reusable in your projects. This documentation includes the board's schematics, OrCAD layout source and Gerber output files, BOM, memory controller CPLD Verilog source code and PDF manuals. The SDK-LITE provides a complete set of Windows host side software and tools, including host-side Windows 98/NT/2000 drivers for the reference

board, PCI 9052 specific APIs and object code libraries, and the PLXMon Windows GUI debug tool. The figure 3.9 shows the RDK-LITE hardware reference board from PLX Technology.

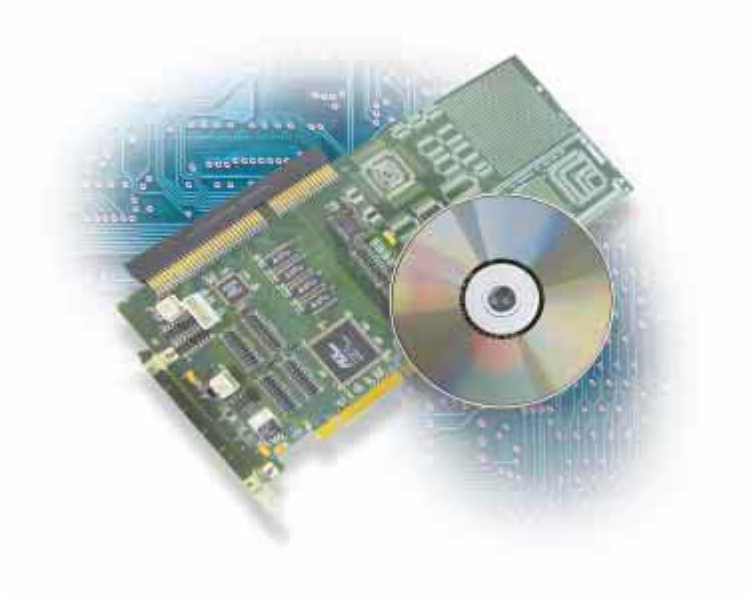


Figure 3.9 PCI 9052RDK-LITE hardware reference board

3.2.9 QV-PCI32RDK-144 Reference Board from QuickLogic.

The QV-PCI32RDK-144 Reference Board allows for demonstration, testing, and debuging of QuickPCI designs which use 144 pin TQFP packages. QuickPCI devices that use this package include the QL5030 (target) and QL5130 (target). Eight LEDs visible from the top edge of the card are provided for testing and to help the evaluation process. Test connectors (AMP 2-767004-2) compatible with Hewlett Packard or Tektronix logic analyzers are provided to give the developer total access to the busses and control signals.

A 32-bit multiplexed address/data bus is provided on the local interface to the QL5X30 (U4). 100-MHz synchronous FIFO (36-bit x 4K deep, U5) is connected to the local bus (in a loop back mode) as well as to a 60-pin connector (J2), which can be used for benchmarking, measurements, connection of a simple add-on card, or debugging. The local bus operates synchronously to the local clock, which is derived from a programmable clock circuit (Quality Semiconductor QS5925, U1). This clock generator derives its clock from a 14.318 MHz crystal, can drive the clock from 28 MHz to 114 MHz with a configurable PLL. The control for the PLL can be driven from the

programmable region of the QL5X30 device. A serial EEPROM socket (U3) and interface is provided to configure the QL5X30 RAM modules on power-up, based on the application. The card form factor is compliant with PCI 2.2, (106.68 x 174.63 mm Short Card). The figure 3.10 shows the diagram of QV-PCI32RDK-144 Reference Board.

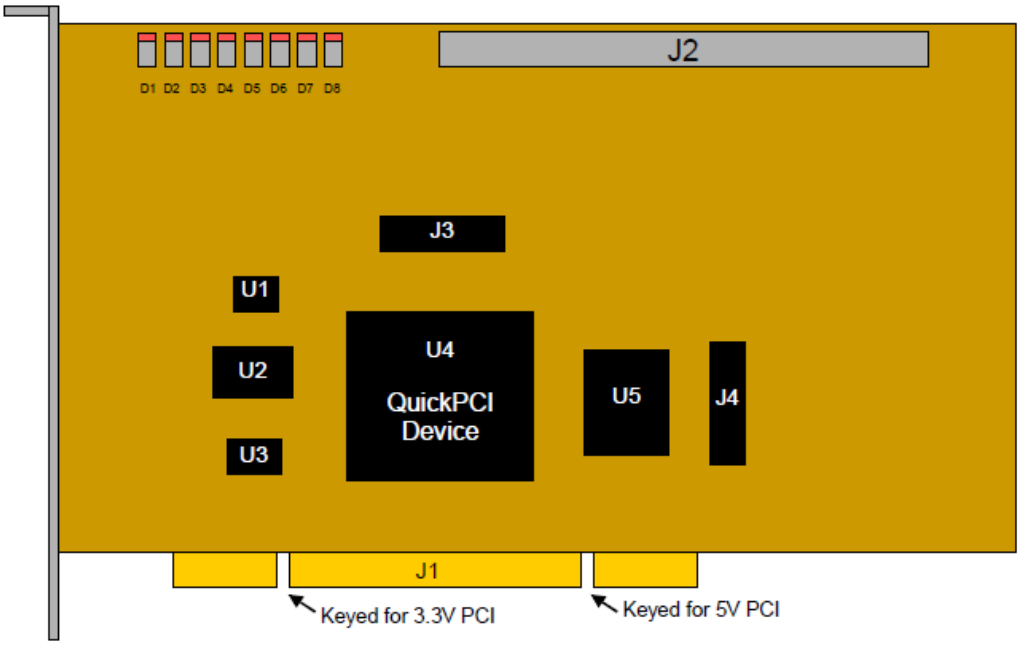


Figure 3.10 PCI32RDK-144 Reference Board

3.2.10 SB16C1052PCI Development Kit from SystemBase Co. Ltd.

SystemBase offers SB16C1052PCI Development Kit to minimize development efforts and costs, and to maximize application stability. SB16C1052PCI is a single chip which enables two asynchronous serial communication ports to be connected to the PCI bus without any glue logic and it's the best solution to constitute a serial port

for the PCI bus. It can enable customer to make easy and simple reference design as a one-chip solution for 1-port or 2-port Serial MultiPort Boards. It includes SB16C1052, Dual UART with 256-Byte TX/RX deep FIFO developed by SystemBase. It also has enhanced features, global interrupt and dedicated control pins for RS422/485 auto toggling. The 256-byte FIFOs reduce CPU overhead and allow higher data throughput. The figure 3.11 shows the SB16C1052PCI Development Kit block diagram.

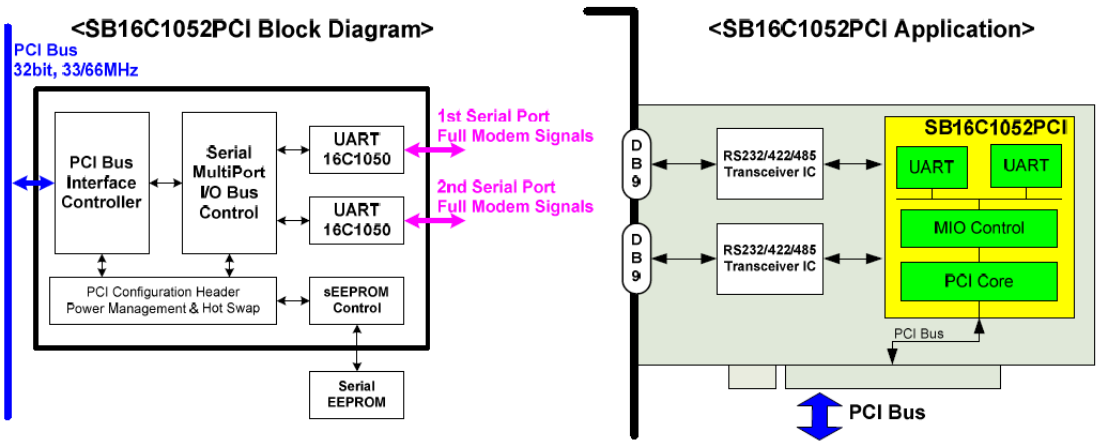


Figure 3.11 diagram of SB16C1052PCI

3.2.11 AMCC's PCI S5335DK Developer Kit

The PCI Developer's kit contains one printed circuit board plus a CD-ROM with software tools. The S5335 PCI card contains an S5335, SRAM, a pre-programmed CPLD containing Add-On bus control functions and a 9 pin, RS-232 connector to communicate with the AddOn Bus. This card was developed to demonstrate interconnection of the S5335 PCI interface chip to the PCI Bus and interconnection of the S5335's Add-On Bus to a basic SRAM design. The onboard CPLD is specifically programmed to control the Add-On bus for Active Mode data transfers for burst or single cycle data reads and writes to the SRAM. It's presently being controlled through the GUI and CPLD.

The Add-On bus signals are also routed to a set of external application connectors. These connectors provide the designer with additional Add-On bus connection capability. The designer can utilize these for attaching his/her own application PCB to the PCI card's Add-On bus. These connectors are designed to provide connection of the user's custom PCB or a logic analyzer.

It is important for the designer to remember, the developer's kit was designed to demonstrate various aspects of S5335 user design. The specific CPLD, Add-On logic components and software were chosen to support multiple application illustrations. Therefore, the device costs and complexity is more than will be necessary for many applications

AMCC's S5335 MatchMaker is a powerful and flexible PCI controller supporting several levels of interface sophistication. At the lowest level it can serve simply as a bus target with modest transfer requirements. For high

performance applications, the S5335 can become the bus master and can attain the peak transfer capabilities of 132 Mbyte/sec with a 32 bit bus. The MatchMaker is an off-the-shelf, low cost standard product that allows a faster time to market by allowing the designer to focus on the actual application and not on debugging the PCI interface/ the figure 3.12 shows the system block diagram of S5335.

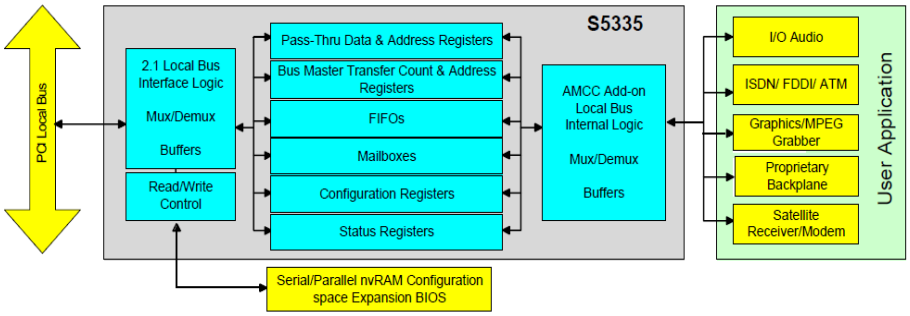


Figure 3.12 System block diagram of S5335

3.2.12 The Cyclone II EP2C35 PCI development board from Altera

The PCI Development Kit, Cyclone™ II Edition is a complete PCI and PCI-X prototyping and testing kit, based on the Cyclone II device. With this kit, you can perform various PCI transactions between the board and the host PC, as well as configure the board with either the factory-programmed or user-programmable design. In addition to providing a PCI/PCI-X form factor development board, the kit also includes all of the hardware and software development tools, as well as the documentation and accessories you need to begin developing PCI systems using the Cyclone II device.

The Kit includes:

- The Cyclone II EP2C35 PCI development board
- *Quartus® II Software, Development Kit Edition* (DKE)
- PCI-to-DDR2 reference design
- MegaCore IP Library CD-ROM

The figure 3.13 show diagram of Cyclone II PCI development kit

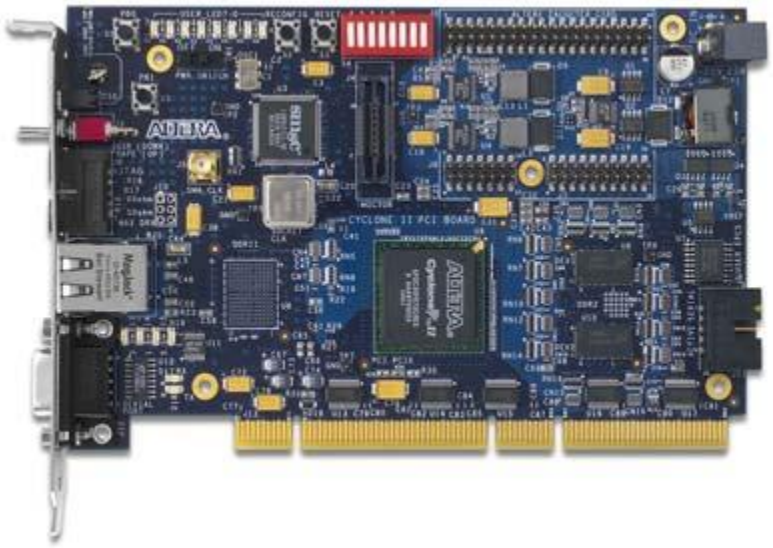


Figure 3.13 Cyclone II EP2C35 PCI development board

3.2.13 MCS98XXCV-BA ASIC Evaluation Board from MosChip Semiconductor

The MCS98XXCV-BA is a PCI based single function I/O Adapter. It has two 16C550 compatible UART channels with two IEEE 1284 compliant parallel port and the ISA style interface to add additional external UARTS. The MCS98XXCV-BA is ideally suited for PC applications, such as high speed COM ports and parallel ports. The MCS98XXCV-BA is available in a 128-pin QFP package. The figure 3.14 shows the block diagram of MCS98XXCV-BA EVB.

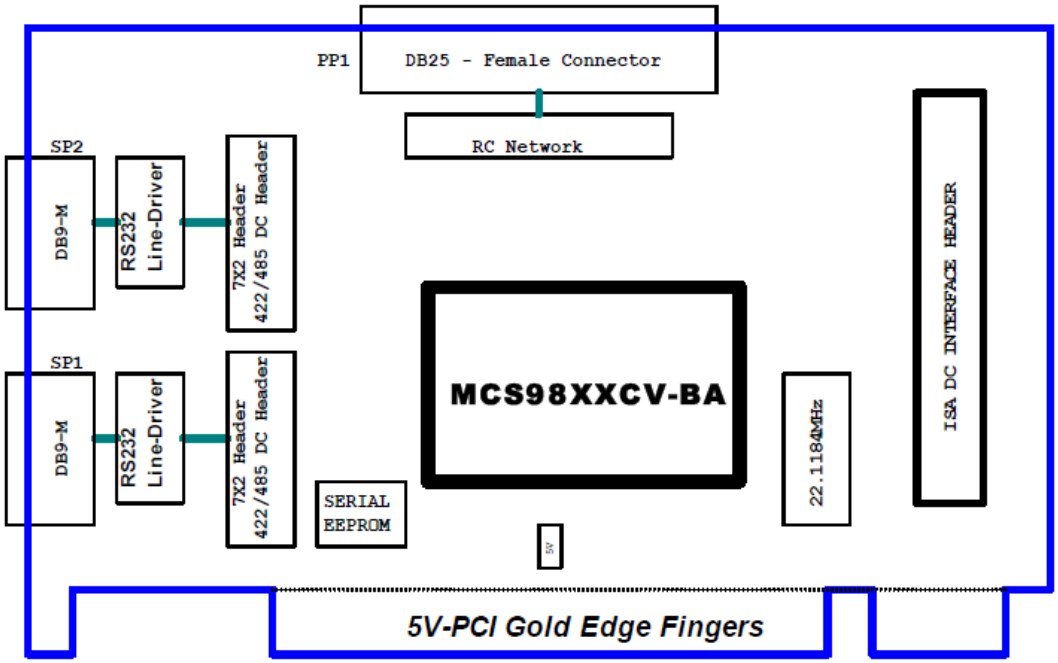


Figure 3.14 Block diagram of MCS98XXCV-BA EVB.

The MCS9835 is a PCI based dual-channel high performance UART with enhanced bi-directional parallel controller. The MCS9835 offers 16-Byte transmit and receive FIFOs for each UART channel and a 16-Byte FIFO for the printer channel. The MCS9835 performs serial-to-parallel conversions on data received from a peripheral device, and parallel-to-serial conversions on data received from its CPU. In addition, MCS9835 fully supports the existing Centronics printer interface as well as PS/2, EPP, and ECP modes.

The MCS9835 is ideally suited for PC applications, such as high speed COM ports and parallel ports. The MCS9835 is available in a 128-pin QFP package. It is fabricated using an advanced submicron CMOS process to achieve low drain power and high-speed requirements. The MCS9835 can be used in both 3.3V and 5V PCI signaling environments. The block diagram of MCS9835CV is show in figure 3.15.

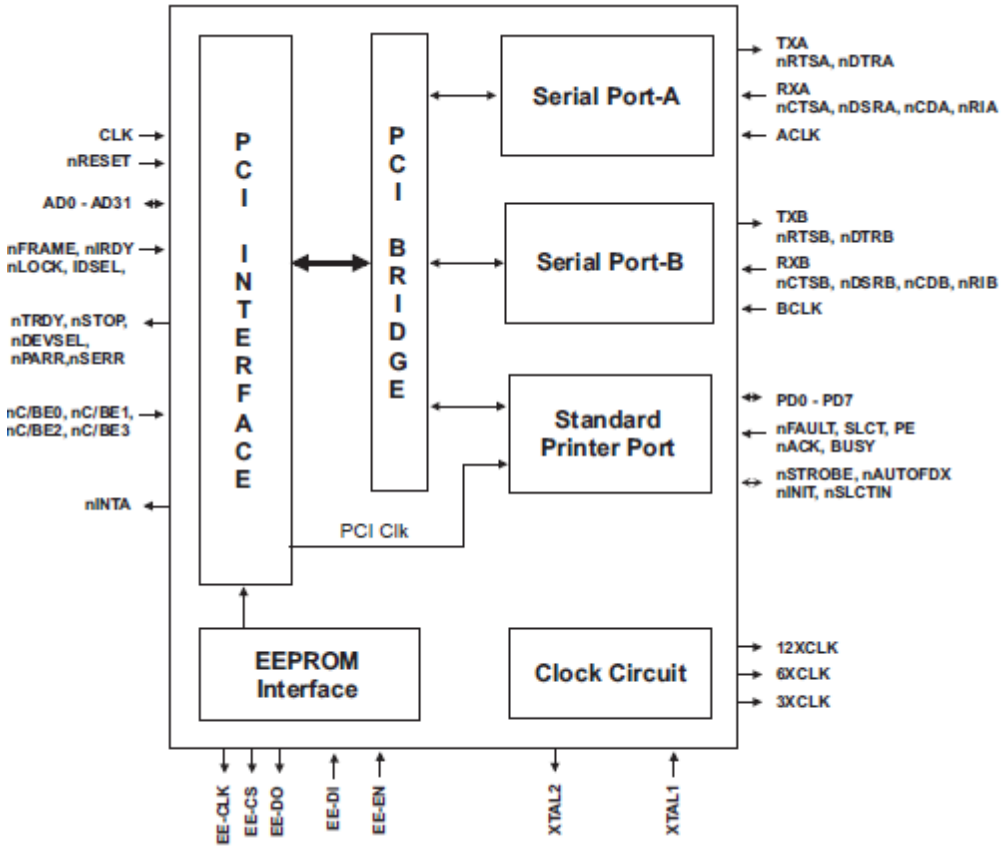


Figure 3.15 block diagram of MCS9835CV

The **MCS9835CV** from **MosChip Semiconductor** is selected as PCI Bus interfacing chip for research work.

3.3 Current Bus system and related work.

It is widely recognized that the computer system bus affects the system characteristics in several important ways:

- The bus bandwidth and transfer parameters place a limit on the system performance.
- The system bus is an interface that connects hardware components produced by different vendors and provides interoperability.
- The wide variety of configuration options supported by increasingly complex and sophisticated I/O devices make manual configuration a difficult and error-prone task. Support for software based automatic configuration has become a necessity.

- When multiple processors share a bus with common resources, some form of support for multiprocessing is required to arbitrate the use of shared resources.

Even though memories are getting faster, CPUs get faster quicker. Although the memory burst speed can be increased by using interleaving, the initial latency cannot be reduced, and in fact becomes the dominant factor in bus usage. This is just one of a number of parameters, other than demand for raw bus bandwidth, that have changed in recent years and must be considered in modern system bus design.

In real-time operating systems for embedded environments, designed to control dedicated hardware such as actuators and sensors, the amount of data handled and the required bandwidth is relatively low. Only a few control messages and some measurement data have to be exchanged between peripheral devices and the main unit. In this case, it is sufficient to use a bus system with support for guaranteed transmission latency; bandwidth scheduling is not an issue here. Systems with a demand for higher bandwidth such as aerospace and avionics control, robotics, medical image processing, or video-on-demand servers use bus systems with explicit real-time support such as the VMEbus[8].

In the recent past, off-the-shelf PC systems for the consumer market—in general with the PCI bus— have become so powerful that “real-time processing” is an often-heard buzzword. Many applications, especially in the scope of audio and video processing, are available. However, such applications often only claim real-time capabilities but do not even run on a real-time operating system. If real-time systems are used, bus bandwidth is completely disregarded as a resource. One reason for this is the assumption that, merely by using powerful hardware, bus bandwidth is always available in the requested amount and is limited only by the hardware-given maximum bandwidth, and that there is never any shortage of this resource. Another reason why bus bandwidth is not seen as an “operating-system manageable” resource is the large difference in the timing base. While buses operate in ranges of nano to microseconds, the time base of CPU scheduling is in the range of tens to hundreds of milliseconds. To our knowledge, no real-time operating system for standard hardware, neither in the research community

nor in the commercial arena, deals with bus bandwidth as a resource to be managed. Some support is given for external bus systems that provide bandwidth guarantees. However, support is restricted to the directly provided capabilities of the hardware. For example, drivers for the Universal Serial Bus (USB) support bandwidth guarantees for data transfer between USB devices and the USB host adapter (since this is supported by the physical USB hardware), but do not guaranteed that enough bandwidth is also available and reserved on the system's local bus to finally transfer the data into main memory. Small operating systems on predictable hardware adapter cards handle bandwidth to guarantee given commitments for this hardware, but not to manage the local buses of the card.

It is interesting to review the way microprocessor system buses have evolved beginning with the ISA[9] and up to new modern buses, such as PCI, FutureBus+, and VME64, and how various tradeoffs and compatibility issues were addressed in each design. Various bus systems are as follows.

1. Industry Standard Architecture.
2. VersaModule Eurocard Bus.
3. FutureBus and FutureBus+.
4. InfiniBand.
5. Controller Area Network.
6. The Cambridge Desk Area Network.
7. Universal Serial Bus.
8. IEEE-1394 or Firewire.
9. Accelerated Graphics Port.
10. EISA.
11. PCI.

In this section we give a short overview of various bus systems and their behavior regarding arbitration and timing constraints. These buses are listed in this section because they are either used in real-time systems, provide capabilities for bandwidth reservation, or have influenced design decisions of the PCI bus. Table 3.1 summarizes the bus width, clock frequency, and maximum bandwidth of common bus systems.

Table 3.1.: Parameter comparison of common bus systems

Bus Name	Width (bits)	Frequency (MHz)	Max. Bandwidth (MB/s)
8-bit ISA	8	8.3	8.3
16-bit ISA	16	8.3	16.6
EISA	32	8.3	33.2
VLB	32	33	132
32-bit PCI	32	33	132
64-bit PCI 2.1	64	66	528
AGP	32	66	264
AGP (x2 mode)	32	66 * 2	528
AGP (x4 mode)	32	66 * 4	1056
USB 1	1(serial)	N/A	0.1875(sync) 1.5(async)
USB 2	1 (serial)	N/A	40
FireWire / IEEE-1394	1 (serial)	N/A	600
FutureBus	32	100	400
FutureBus+	265	100	3200
VMEbus	64	10(1000nS cycle)	80
CAN bus	1 (serial)	Variable	Len > 100m: $\frac{75}{bus\ len(m)}$

3.3.1. Industry Standard Architecture Bus

The ISA Bus [3, 9] originated in the IBM PC, introduced in 1981. In it's first version, the ISA Bus was only 8 bit wide, with a 1Mbyte addressing range. When the IBM AT was introduced, another connector was added along the original XT Bus connector, adding 8 additional data lines, 4 additional address lines, and more interrupt/DMA lines. There was no organization defining the ISA bus, so when more AT compatible machines appeared on the market, the ISA Bus became a de-facto standard. Only in a later stage the IEEE defined the ISA bus as IEEE 996, but this was done as an afterthought. In fact, the objective of new systems having ISA slots is to be compatible with as many ISA cards as possible, IEEE compliant or not. Later on, Plug and Play was

added to ISA bus. There are even a few non x86 based machines using the ISA bus (some SGI machines).The maximum transfer rate of the original version is 8.3MB/s at a clock speed of 8.3MHz. This provides reasonable throughput for devices with low bandwidth requirements. All transactions are actively controlled either by the CPU or a single DMA controller. The CPU and the DMA controller synchronize their bus accesses using a direct wire. All devices are “passive,” that is, they cannot actively request the bus and transfer data to main memory autonomously. Hence, an arbitration process is not required. However, devices can be programmed to announce available data to the DMA controller which then performs the transfer without further CPU interactions. Since the DMA controller is only programmed by the CPU, no uncontrolled data transfer between a device and the main memory can occur. This makes the bus very predictable but does not provide the performance required for multimedia applications. In current systems, the ISA bus is usually connected via a PCI-to-ISA bridge and is present only to support legacy devices. The ISA bus will be removed from systems at some point in the future.

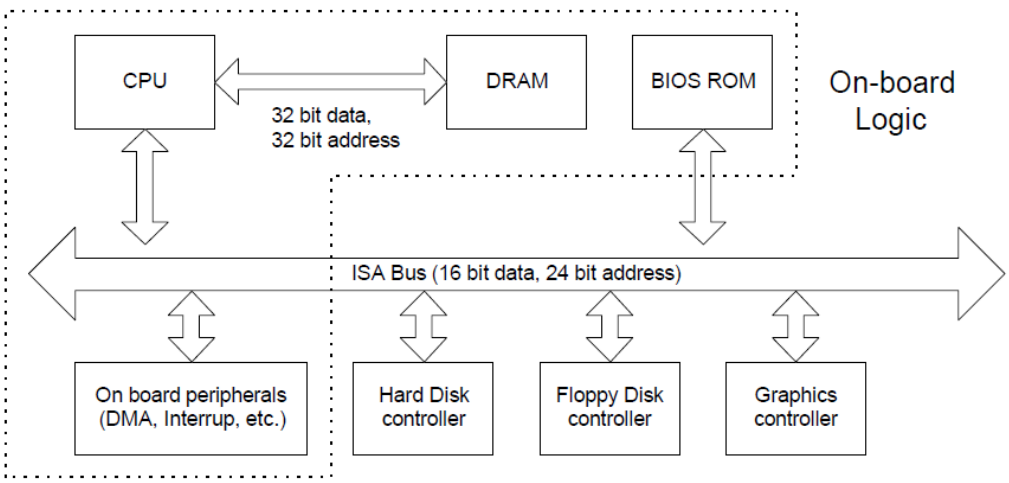


Figure 3.16 ISA Architecture

3.3.2 VersaModule Eurocard Bus

VME [9] is the standard bus type on high performance multiprocessor servers, and high end embedded systems. VME began it's life as a 16 data bit/24 address bit bus for 680X0 based machines called VERSAbus. VERSAbus

was later modified to use the EuroCard form factor, and was renamed to VME (Versa Module Eurocard). Today VME supports 32 data bit/32 address bit cards, and VME64 supports 64 bit cards. VME system supports all popular microprocessors today, including the 680X0, the SPARC, the Alpha, and the X86.

The VersaModule Eurocard (VME) bus is an asynchronous, master-slave, parallel bus. The original bus is 32 bits wide and supports transfer rates up to 40MB/s, current systems are 64 bits wide and provide a bandwidth of 80MB/s. Recent versions (2eSST) of VMEbus achieve a sustained backplane performance of 1GB/s. VMEbus is mainly used in embedded systems, such as factory automation, avionics, in-flight video servers, cellular-phone base stations, and many others.

Every VMEbus must have a system controller, which performs the arbitration. The VMEbus system controller must reside in slot one and provides its functionality independent of the card plugged into that slot. Hence, a master or a slave may coexist with the system controller in the same slot. Three arbitration modes are available: priority mode (PRI), round robin (RR), and single level (SGL). Selecting the arbitration mode is part of the system initialization process; the arbitration mode cannot be changed while the system is running. Priority mode provides four bus request levels (BRL) and gives highest priority to bus request level three, then two, one and finally zero. For devices at the same request level, proximity to slot one determines which device gets the bus next. Round-robin mode grants the bus in sequential order, i.e., BRL 3, 2, 1, 0, 3, 2, 1, etc. Single-level arbitration honors only BRL 3, other levels are ignored. A mix of PRI and RR mode is also allowed by the VMEbus specification. Here, BRL 3 works in PRI mode, BRL 2, 1, and 0 in RR mode. Alternatively, the bus allows using PRI mode for BRL 3 and 2, but RR mode for BRL 1 and 0. While PRI mode is suitable for systems or devices with real-time requirements, RR mode is designed to support applications where fair sharing of the bus is required. SGL mode is provided for simple systems that do not require a sophisticated arbitration scheme. The basic support for priorities makes VMEbus especially suitable for real-time environments. Since the main target are embedded systems with a relatively static design, priorities can easily be calculated up front.

3.3.3. FutureBus / FutureBus+

FutureBus+ [10] is an asynchronous high-performance bus and an improvement of the old FutureBus. It is architecture and processor independent. The current standard supports bus widths up to 256 bits. The bus supports both centralized and decentralized arbitration. Each module has a unique 8-bit arbitration number. The module that supplies the highest number is granted bus access by the arbitration logic. Data transfer is burst-oriented—after sending a target address, the entire data burst is sent. Currently, FutureBus and FutureBus+ are rarely used, but they have strongly influenced the development of current state-of-the-art bus systems.

An approach for real-time traffic over FutureBus+ based on rate-monotonic theory is given by Sha et al. [11]. This is also the recommended approach in the FutureBus+ System Configuration Manual (IEEE 896.3) [12].

The FutureBus+ standard is split into multiple sections:

· The electrical characteristics of the FutureBus+ signals such as signaling levels and load curves.

· The mechanical section defines the FutureBus+ form factor parameters such as card dimensions and the connector type used.

· The FutureBus+ protocol is the core of the standard, and defines the various signals and their behavior.

The use of separate definitions allows some parts of the standard to evolve, while staying compatible with other aspects of the bus.

Some of the special features of FutureBus+ can be summarized here:

· Architecture, Processor, and technology independence.

· No technology based upper limits. The only speed limiting factor in the standard should be physical limitations (speed of light). The result of this guideline was an asynchronous bus which can go faster as technology improves.

· Fault Tolerance: parity protection on all lines, fully distributed arbitration protocol (to reduce the risk of a single failure point), live insertion and removal of modules, dual bus operation, and fault detection and isolation mechanisms.

· Full support for cache coherency protocols, and split transactions.

· Message passing protocol for efficient multiprocessor communication.

· Full Support for bus to bus bridges. Bridges for the following buses has been defined in the standard: VMEbus, Multibus II and SCI (Scalable Coherent Interface) [13].

One of the major applications of FutureBus+ is multiprocessing, as illustrated in Figure 3.17. The most striking feature in Figure 3.17 is the hierarchy of buses interconnected by a set of bridges. The parallel protocol supports split transactions, required for efficient communications across buses. FutureBus+ integrates a MESI cache coherence protocol, and supports snarfing. While somewhat more complex to implement than a read invalidate protocol, snarfing allows cache-to-cache transfers of modified cache blocks without updating the memory. Snarfing can save significant bus bandwidth as it performs a single transfer of the modified cache block to the requester, rather than two transfers (memory write of the modified block followed by a memory read by the requester) in the simpler read invalidate protocol.

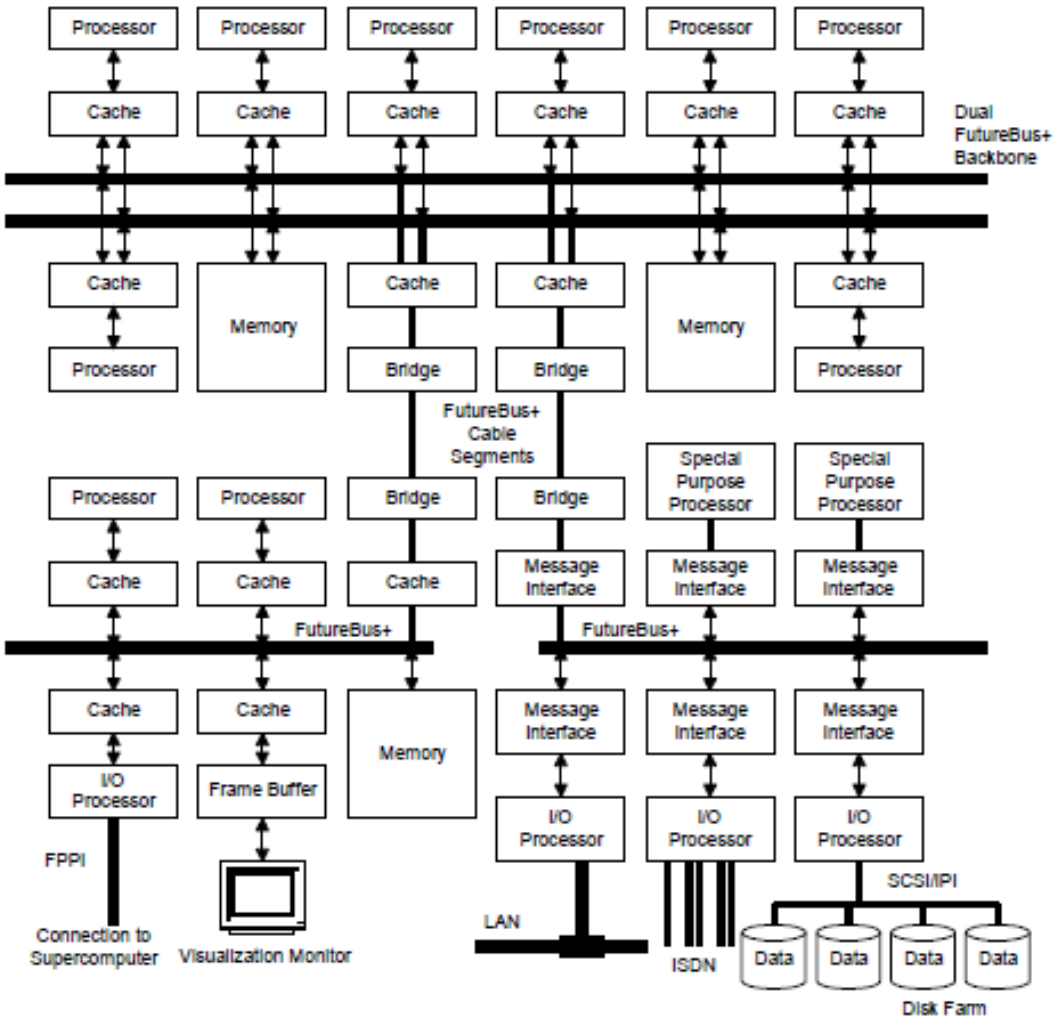


Figure 3.17 FutureBus+

3.3.4. InfiniBand Architecture

The InfiniBand Architecture [14] specifies a “first-order interconnect technology,” i.e., a system for connecting processors and I/O nodes to build a system area network. The architecture is independent of the host operating system and the processor platform. The InfiniBand architecture is based on a point-to-point, switched I/O fabric. All end-node devices are connected via switch devices. InfiniBand is designed to become the backbone for server systems with support for bandwidth and latency guarantees. Similar to ATM networks, a certain amount of bandwidth is reserved per connection, and the switch fabrics are responsible for guaranteeing and enforcing these reservations. These features make InfiniBand an interesting alternative for

future real-time systems. At this time, information is very preliminary and not publicly available.

3.3.5. Controller Area Network Bus

The Controller Area Network [15] is a serial communication protocol that supports distributed real-time control systems. Originally designed for connecting components in automotive systems, it has applications in many industrial automation and process-control systems because it is a simple and low-cost system. The CAN bus allows real-time and non-real-time applications to share the network. The bus-access protocol is based on the carrier sense multiple access / non-destructive bitwise-arbitration scheme (CSMA/NBA). After sending a preamble to indicate the start of a transmission, a priority identifier is sent. Each node continuously senses the bus and checks its state. Each node compares the scanned state on the bus with the state it has put on the bus itself. In case of a mismatch, the device drops out of connection. Finally, the sender transmitting the highest identifier hereafter gains exclusive access to the bus. To avoid performance degradation, this mechanism requires an efficient control mechanism to provide guaranteed access for real-time applications. Bus scheduling based on earliest-deadline-first (EDF) techniques has been described elsewhere [16, 17]. Compared to static-priority scheduling such as rate-monotonic scheduling or deadline-monotonic scheduling, the proposed approach allows an increase of up to 20% in the feasible network workload. An approach that combines both EDF and fixed-priority scheduling is given by Zuberi and Shin [17].

3.3.6. Cambridge Desk Area Network

The Cambridge Desk Area Network (DAN) [18] uses ATM techniques to build a multiprocessor multimedia station around an ATM switch. Various intelligent autonomous components (also called nodes) such as CPU, frame grabber, network cards or independent memory/storage nodes are connected via the DAN. Due to its switching technology, this system scales very well by connecting multiple DANs with each other. The computing nodes, input/output nodes and even the integrated parts of a typical desktop station can be separated. Since it also serves as a local area network, personalized I/O nodes can travel along with the user while the computation nodes always stay connected and online.

By design, the cell-based technology of ATM supports bandwidth reservation; hence additional technologies are not required. Once a “channel” between source and destination has been established under a given quality-of-service parameter set, the switches enforce this service quality.

3.3.7. Universal Serial Bus

Universal Serial Bus (USB) [19] is a serial bus designed to overcome known problems of low- to mid-speed buses, such as inflexibility, limited scalability and extensibility, or missing plug-and-play capabilities. With USB-1, a well-defined interface for hardware and software interconnection allows an easy integration of new devices into current systems. The first version supports bandwidths up to 12Mbps for asynchronous or 1.5Mbps for synchronous data transmission. The increasingly higher performance and capability to process huge data streams in end systems also results in a need for higher speed and bandwidth in the interconnect and bus systems. This is the motivation for USB-2 [19], which now supports transfer rates of up to 480Mbps. USB devices are distinguished based on their bandwidth consumption in low-speed (10–100kbps), full-speed (500kbps–10Mbps), and high-speed (25-480Mbps) devices. All devices are connected point-to-point to a hub device. The hub device itself can be connected to another hub, building a hierarchy as illustrated in Figure 3.18.

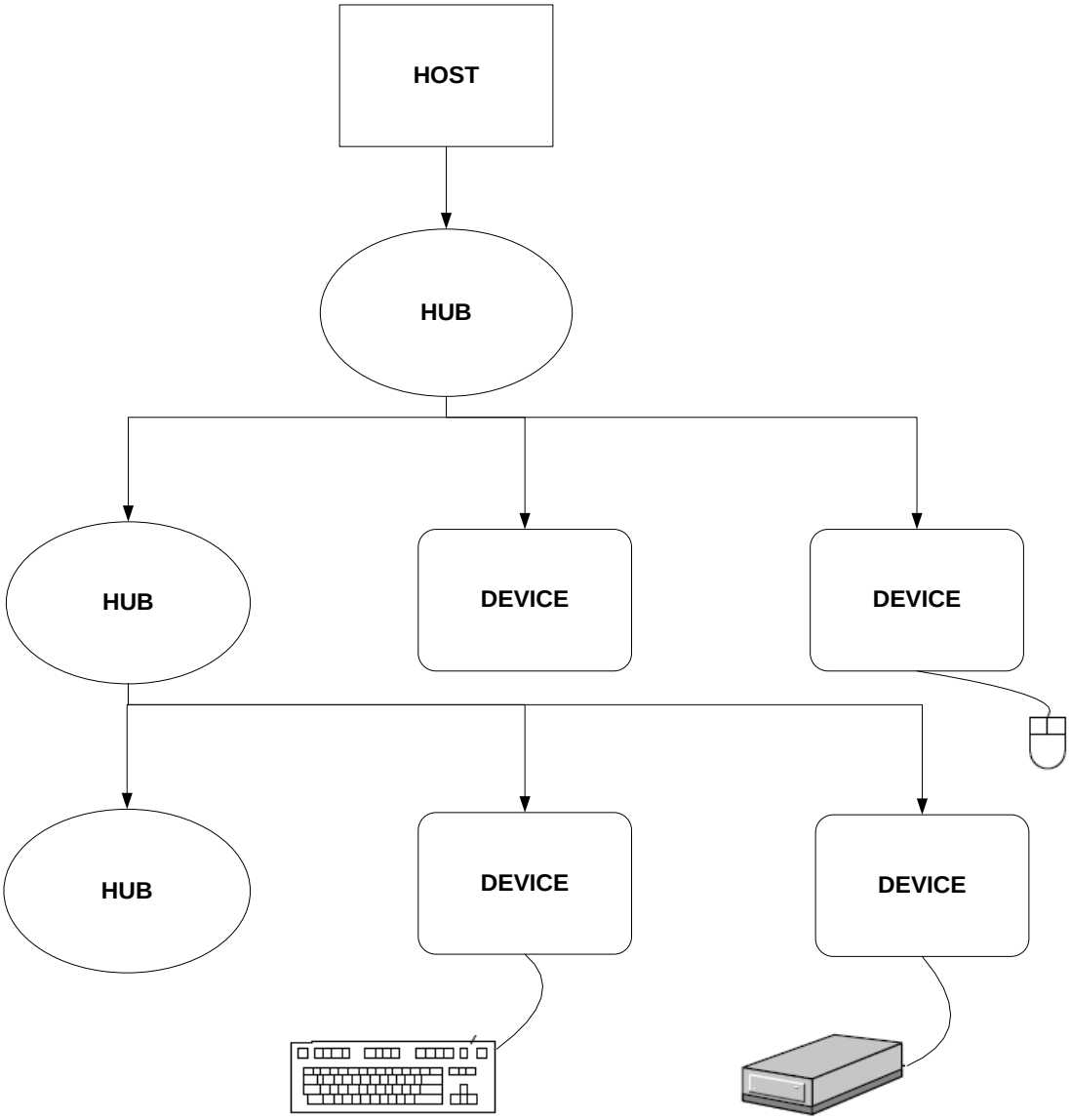


Figure 3.18 USB device hierarchy

All transfers are based on pipes, a point-to-point association between two endpoints, the USB host and the USB device. Each pipe holds information about the transfer type and the requirements of the stream. To support various requirements of devices, USB provides the following transfer types: control transfer, interrupt-data transfer, bulk-data transfer, and isochronous-data transfer. Control transfers are mainly used to configure a device at the time it is attached to the bus but can also be used for other device-specific purposes. The bulk-data transfers have wide dynamic latitude in transmission constraints and are generated or consumed by devices in relatively large and bursty quantities. Interrupt-data transfers are used for timely critical (low

latency) and reliable delivery of data, such as keyboard or mouse events. Isochronous data transfers are targeted to transfer real-time data with constant bit rate. They occupy a prenegotiated amount of USB bandwidth with prenegotiated delivery latency; bandwidth and latency are guaranteed by the bus. Time on the bus is divided either into 1ms frames for full-speed or 125µs micro-frames for high-speed segments. Isochronous and interrupt endpoints are given opportunities to access the bus every N (micro)frames. Isochronous transmissions are guaranteed to occur within a prenegotiated frame but may occur anywhere within the frame.

3.3.8. IEEE-1394 Serial Bus / FireWire

The standard described in IEEE-1394 and IEEE-1394A, also known as FireWire or iLink, is a high-performance serial bus targeting high-speed computer peripherals such as video cameras, audio devices, network interfaces, etc.; it also supports low-speed, non-time-critical devices such as printers and hand-held devices. The design and application area of IEEE-1394 is similar to that of USB. Data rates up to 400Mbps are currently supported. Future plans to support data rates of 3,200Mbps would allow using IEEE-1394 buses as the common backplane within hosts. The main focus of the IEEE-1394 serial bus is an easy connection of external devices to a host. Similar to USB, easy hot-plug-and-play technology provides the ability to add or remove devices at any time. Devices are addressed by a 64-bit ID that is partitioned into 10 bits for the network ID, 6 bits for the node ID, and 48 bits for memory addressing within each node. Hence, this topology can support up to 1023 networks or buses of 63 devices per bus (node). The address space each individual device can address is 281 terabytes. This direct addressing mode of IEEE-1394 is the major advantage over USB. Devices can be directly connected “back-to-back” without additional hardware. From a device’s perspective, data connectivity is a true peer-to-peer connection since each device’s memory is addressable from every point in the topology.

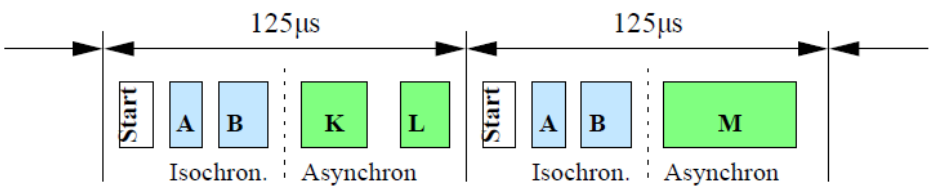


Figure 3.19 IEEE-1394 bus cycle with isochronous and asynchronous data

In contrast to PCI, where only the electrical mechanism and timing constraints for arbitration are defined, IEEE-1394 also defines the arbitration scheme. In their technical construction, IEEE-1394 and USB are very similar. Time on the bus is divided into 125 µs frames. Figure 3.19 illustrates a typical cycle on the bus. To ensure bandwidth guarantees, up to 80 % of the total bandwidth can be reserved for isochronous (real-time) data streams. The rest is available for non-real-time data transfers, called asynchronous data. A variable-length gap between data packets is an essential part of the arbitration algorithm and indicates the maximum propagation delay. This is required since IEEE-1934 packets are not equally sized within a cycle and bus participants recognize a packet end by means of such a gap. Gaps between isochronous packets are shorter than those between asynchronous packets.

3.3.9. Accelerated Graphics Port

The Accelerated Graphics Port [20] (AGP) is a point-to-point connection between the graphics controller and the main memory. It is used exclusively to connect display adaptors and graphics devices; the primary intention was to improve graphics performance. By providing a real bandwidth improvement between the graphics accelerator and main memory, some of the data structures used for 3D graphics may be moved to main memory. This effectively reduces the costs for accelerated graphics adaptors. Texture data are especially suited to a shift to main memory, since they are generally read-only and therefore do not have special access ordering or coherency problems.

AGP is, in general, based on and derived from the 66MHz PCI-bus specification but provides significant performance extensions such as deeply pipelined memory and de-multiplexing of address and data on the bus. Since data transfers are possible at both falling and rising edge of a clock cycle, the transfer rate is twice as high as for the PCI bus. Given its typical point-to-point connection topology, bandwidth guarantees exist automatically and no contention or conflicts on the AGP can arise. Contention is possible in the host bridge where AGP and the PCI bus are connected to the memory bus. From the bridge point of view, an AGP device has the same properties as an entire PCI-bus subsystem connected to the bridge.

3.3.10 EISA

The EISA architecture is quite similar to the MicroChannel architecture shown in Figure 3.20. To remain compatible with ISA, EISA allows the use of ISA cards, and also retains the maximum 8.33 MHz clock rate. The EISA bus controller (EBC) supports 32- and 16-bit EISA adapters, and 16- and 8-bit ISA adapters. Transfers may occur between ports with different widths. For example, assume a 32-bit EISA bus master performs a write to an 8-bit ISA slave. The bus master gains access to the bus

and drives address and data signals on the bus. The 8-bit ISA slave indicates it can only perform 8-bit transfers. At this point, the EISA bus controller takes over and drives on the bus the same signals as the bus master. Then the bus master deactivates its drives, and the signals, now driven only by the bus controller, remain steady on the bus. Finally, the bus controller splits the 32-bit data into four ISA byte writes. Read accesses are accumulated and combined into 32-bit reads in a similar way. The EBC implements the logic that splits/combines data and generates multiple access cycles, and bus masters are not required to duplicate it.

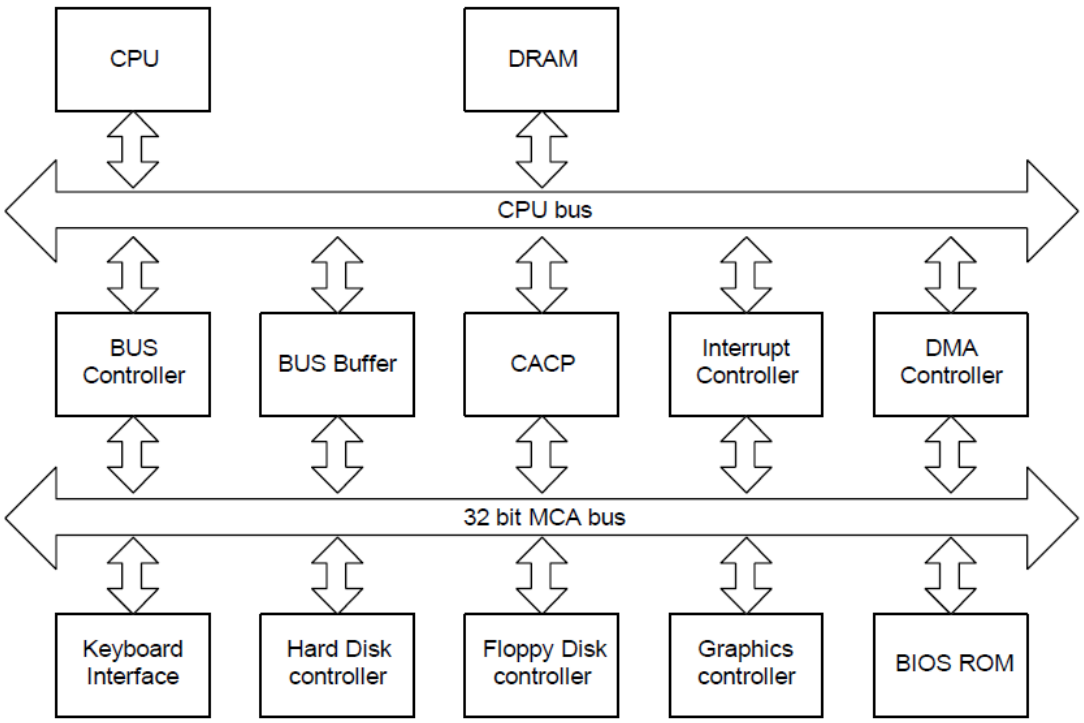


Figure 3.20 MCA Bus

EISA supports burst cycles. There is also a more recent specification of an enhanced burst cycle, that provides higher transfer rates while keeping the bus clock at 8.33 MHz for compatibility. A 66Mbyte/s rate is achieved by performing 32-bit burst transfers on both edges of the clock. A 133Mbyte/s rate is achieved by multiplexing the 32-bit address bus to transfer 64 bits (using both the address and data buses), on both edges of the clock.

A common problem in ISA machines is the lack of interrupt lines. EISA solves this problem by allowing interrupt levels to be level-sensitive, instead of edge-triggered. Multiple devices may share the same interrupt line. While one device is serviced, the interrupt line remains active indicate pending interrupts from other devices. ISA adapters are edge-triggered, and require separate interrupt lines even if plugged into an EISA bus. Only multiple EISA adapters can share the same line.

3.3.11 PCI

PCI is a *local* bus, sometimes also called an intermediate local bus, to distinguish it from the CPU bus. The concept of the local bus solves the downward compatibility problem in an elegant way. The system may incorporate an ISA, EISA, or MicroChannel bus, and adapters compatible with these buses. On the other hand, high-performance adapters, such as graphics or network cards, may plug directly into PCI (see Figure 3.21). PCI also provides a standard and stable interface for peripheral chips. By interfacing to the PCI, rather than to the CPU bus, peripheral chips remain useful as new microprocessors are introduced. The PCI bus itself is linked to the CPU bus through a PCI to Host bridge [21].

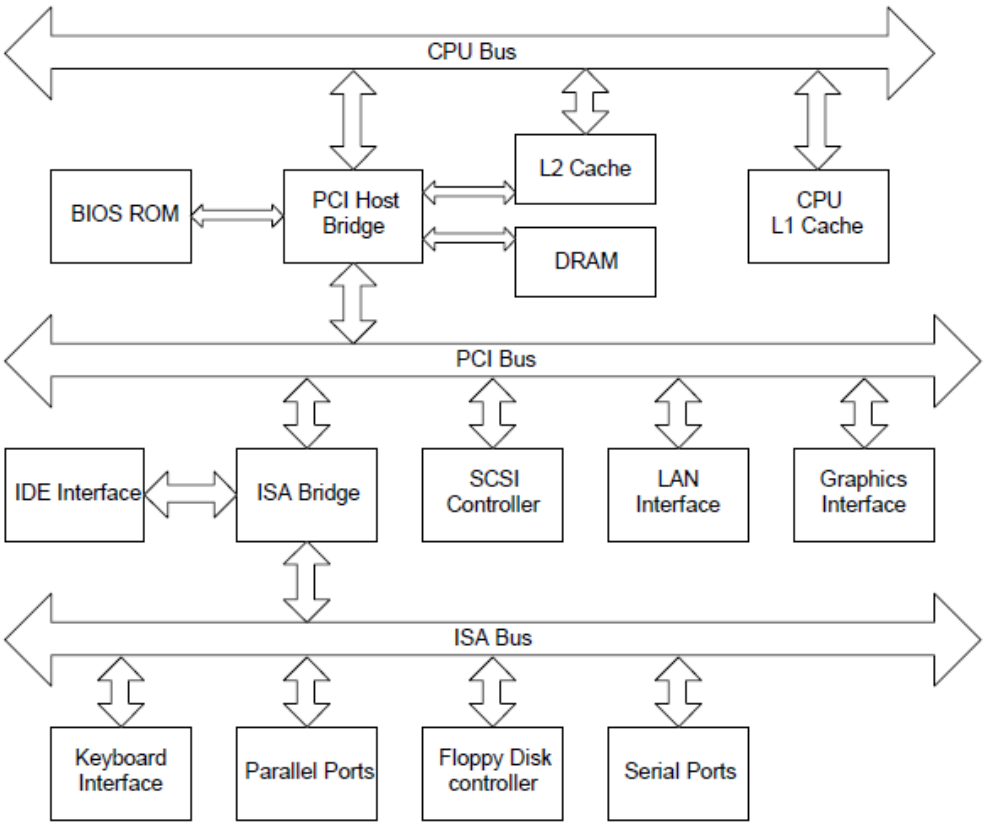


Figure 3.21 PCI Architecture

The basic PCI transfer is a burst. This means that all memory space and I/O space accesses occur in burst mode; a single transfer is considered a “burst” terminated after a single data phase. Addresses and data use the same 32-bit, multiplexed, address/data bus. The first clock is used to transfer the address and bus command code. The next clock begins one or more data transfers, during which either the master, or the target, may insert wait cycles. PCI supports *posting*. A posted transaction completes on the originating bus before it completes on the target bus. For example, the CPU may write data at high speed into a buffer in a CPU-to-PCI bridge. In this case the CPU bus is the originating bus and PCI is the target bus. The bridge transfers data to the target (PCI bus) as long as the buffer is not empty, and asserts a not ready signal when the buffer becomes empty. In the other direction, a device may post data on the PCI bus, to be buffered in the bridge, and transferred from there to the CPU via the CPU bus. If the buffer becomes temporarily full, the bridge deasserts the target ready signal.

In a read transaction, a turnaround cycle is required to avoid contention when the master stops driving the address and the target begin driving the data on the multiplexed address/data bus. This is not necessary in a write transaction, when the master drives both the address and data lines. A turnaround cycle is required, however, for all signals that may be driven by more than one PCI unit. Also, an idle

clock cycle is normally required between two transactions, but there are two kinds of back-to-back transactions in which this idle cycle may be eliminated. In both cases the first transaction must be a write, so that no turnaround cycle is needed, the master drives the data at the end of the first transaction, and the address at the beginning of the second transaction. The first kind of back-to-back occurs when the second transaction has the same target as the first one. Every PCI target device must support this kind of back-to-back transaction. The second kind of back-to-back occurs when the target of the second transaction is different than the target of the first one, and the second target has the Fast Back-to-Back Capable bit in the status register set to one, indicating that it supports this kind of back to-back.

To reduce data transfer time on PCI, a bridge may combine, merge, or collapse data into a larger transaction. *Combining* refers to converting sequential memory writes into a single PCI burst transaction. For example, a write sequence of (32-bit) double words 1, 3, and 4 can be combined into a burst with four data phases, the second data phase having the byte enables off. Transactions, whose order is not sequential, for example 4, 3, and 1, must remain as separate transactions.

Merging refers to converting a sequence of memory writes (bytes or 16-bit words) into a single double word. Unlike combining, merging can occur in any order. For example, bytes 1, 0, and 3 can be merged into the same double word with byte enables 0, 1, and 3 asserted.

For arbitration, PCI provides a pair of request and grant signals for each PCI unit, and defines a central arbiter whose task is to receive and grant requests, but leaves to the designer the choice of a specific arbitration algorithm. PCI also supports *bus parking*, allowing a master to remain bus owner as long as no other device requests the bus. The default master becomes bus owner

when the bus is idle. The arbiter can select any master to be the default owner.

Four interrupt lines are connected to each PCI slot. A multifunction unit may use all four, other units may use only a designated line (one of the four). The value in the Interrupt Line register determines which IRQ should be activated by the interrupt signal of the PCI unit. (For example, IRQ14 of the AT architecture, if the PCI unit is a disk adapter). The BIOS fills this field during system initialization.

Later, when the operating system boots, the device driver reads this field to find out to which interrupt vector the driver's interrupt handler should be bound to. PCI provides a set of configuration registers collectively referred to as "configuration space." By using configuration registers, software may install and configure devices without manual switches and without user intervention. Unlike the ISA architecture, devices are relocatable - not constrained to a specific PCI slot. Regardless of the PCI slot in which the device is located, software may bind a device to the interrupt required by the PC architecture. Each device must implement a set of three registers that uniquely identify the device: Vendor ID (allocated by the PCI SIG), Device ID (allocated by the vendor), and Revision ID. The Class Code register identifies a programming interface (SCSI controller interface, for example), or a register-level interface (ISA DMA controller, for example).

As a final example, the Device Control field specifies whether the device responds to I/O space accesses, or memory space accesses, or both, and whether the device can act as a PCI bus master. At power-up, device independent software determines what devices are present, and how much address space each device requires. The boot software then relocates PCI devices in the address space using a set of base address registers. The figure 3.22 shows the different PCI slots in a standard PC.

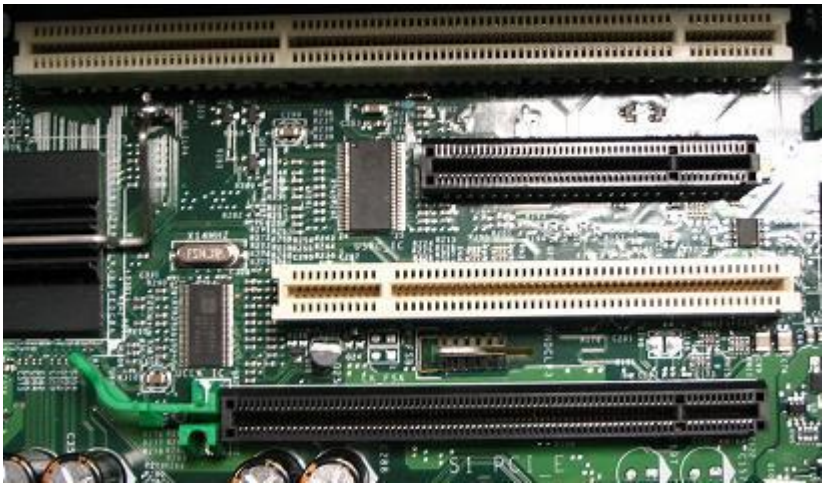


Figure 3.22 Motherboard with Four Slots – PCIe x16, PCI, PCIe x8, and PCI-X (from bottom to top)

3.4 IDE for Microcontroller based system development.

IDE is Integrated Development Environment that is common name for the man –machine software interface for a specific user product. IDE are command line as well as graphical user interface (GUI). With respect to embedded/microcontroller based system include

- Programmer text Editor.
- Assembler and C/C++ compiler.
- Source level debugger for both assembly and C/C++.
- Simulator.
- Support for Add-on software and hardware component, e.g. In-Circuit Emulator/Debugger and programmer.

Different families of microcontroller have respective IDE, some of the well known IDE are described here.

3.4.1 Microchip's MPLAB Integrated Development Environment.

MPLAB Integrated Development Environment (IDE) is a free, integrated toolset for the development of embedded applications employing Microchip's PIC® and dsPIC® microcontrollers. MPLAB IDE runs as a 32-bit application on MS Windows®, is easy to use and includes a host of free software components for fast application development and super-charged debugging. MPLAB IDE also serves as a single, unified graphical user interface for additional Microchip and third party software and hardware development tools. Moving between tools is a snap, and upgrading from the free software

simulator to hardware debug and programming tools is done in a flash because MPLAB IDE has the same user interface for all tools

MPLAB IDE features include:

1 Flexible customizable programmer's text editor.

i. **Fully integrated debugging** with right mouse click menus for breakpoints, trace and editor functions.

ii. **Tabbed editor** option or separate source windows **Recordable macros.**

iii. **Context sensitive color highlighting** for assembly, C and BASIC code readability.

iv. **Mouse over variable** to instantly evaluate the contents of variables and registers.

v. **Set breakpoints and tracepoints** directly in editor to instantly make changes and evaluate their effects.

vi. **Graphical project manager.**

vii. **Version control support** for MS Source Safe, CVS, PVCS, Subversion.

2 **Free components**

i. **Programmer's text editor.**

ii. **MPLAB SIM**, high speed software simulator for PIC and dsPIC devices with peripheral simulation, complex stimulus injection and register logging **Full featured debugger.**

iii. **MPASM™** and **MPLINK** for PIC MCUs and dsPIC DSC devices.

iv. **HI-TECH C PRO for PIC10/12/16 MCU Families** running in lite mode.

v. **CCS PCB C Compiler.**

vi. **Labcenter Electronic's Proteus VSM** spice simulator.

vii. **Many Powerful Plug-Ins** including:

○ AN851 Bootloader programmer.

○ AN901 BLDC Motor Control Interface.

○ AN908 ACIM Tuning Interface.

○ KeeLoq support.

○ Data Monitor and Control

3 Built in support for hardware and add-on components.

i. **MPLAB C Compilers** (free student editions available for download). **MPLAB REAL ICE™** in-circuit emulator.

ii. **MPLAB ICD 2 and MPLAB ICD 3** in-circuit debuggers and engineering programmers for selected Flash devices.

iii. **PICkit 2 and PICkit 3 Debug Express** economy debug/programmers.

iv. **PICSTART Plus** development programmer.

v. **MPLAB PM3** device programmer.

vi. **Third Party tools**, including HI-TECH, IAR, Byte Craft, B. Knudsen, CCS, Micrium, microEngineering Labs, Labcenter, MATLAB, Segger.

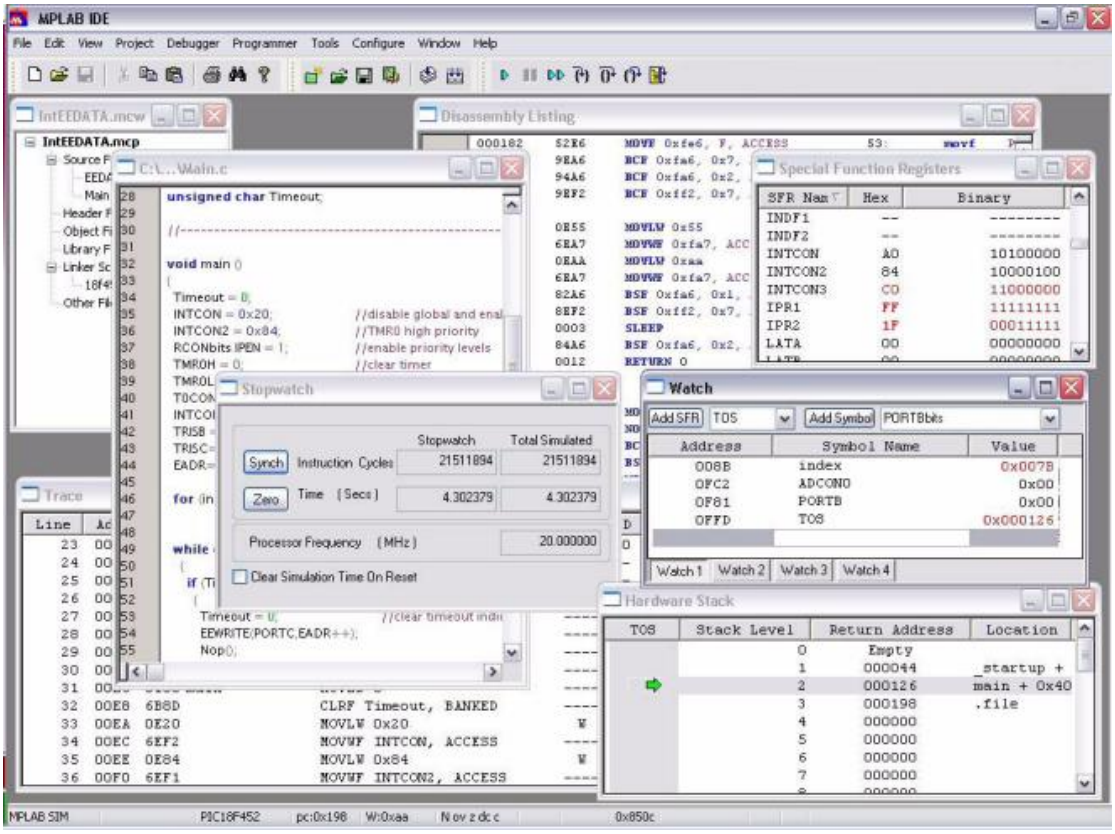


Figure 3.23 MPLAB IDE

MPLAB IDE v8.56 is the latest version of its release from Microchip.

3.4.2 IAR Embedded Workbench®

Efficient and reliable code is a must in an embedded application. IAR Embedded Workbench is the world-leading C/C++ compiler and debugger tool suite for applications based on 8-, 16-, and 32-bit MCUs.

IAR Embedded Workbench is a set of development tools for building and debugging embedded applications using assembler, C and C++. IAR Embedded Workbench provides a completely integrated development environment including a project manager, editor, build tools and debugger. In a continuous workflow, you can create source files and projects, build applications and debug them in a simulator or on hardware.

Benefits

IAR Embedded Workbench offers the same intuitive user interface regardless of which microcontroller you have chosen to work with—coupled with general and target-specific support for each device. Every IAR C/C++ Compiler contains both generic global optimizations as well as low-level chip-specific optimizations that ensure a small code size while taking advantage of all the specific features of your selected device. Reuse of code and migration to new microcontroller architectures is made easy as each IAR C/C++ Compiler uses the same naming convention. Whether you have a tight project schedule or are just eager to get started, IAR Embedded Workbench contains configuration files, code examples and template projects to get you going fast.

Key components

- Integrated development environment with project management tools and editor
- Highly optimizing C and C++ compiler (*)
- Configuration files for all supported devices
- C-SPY® simulator and hardware debugger systems
- Support for RTOS-aware debugging on hardware
- Run-time libraries
- Relocating assembler
- Linker and librarian tools
- Ready-made code and project examples for supported evaluation boards
- User and reference guides in PDF format
- Context-sensitive online help

Integrated development environment

- Hierarchical project presentation
- Multiple projects within the same workspace
- Dockable windows and multiple views
- Source browser
- Library tools included for creating and maintaining libraries
- Integration with source code control systems
- Text editor
- Code templates for commonly used code constructs
- Command line build utility

IAR C/C++ Compiler

- Advanced global and processor-specific optimizations for speed and memory footprint.
- Extended keywords for defining data/functions and declaring memory and type attributes.
- Pragma directives for controlling compiler behavior such as memory allocation.
- Intrinsic functions for direct access from C code to low-level processor operations.
- Full support for memory attributes in C++ (*).
- Support for interrupt and exception handling in C and C++ (*).

IAR Assembler

- A powerful relocating macro assembler with a versatile set of directives and operators
- Built-in language preprocessor, accepting all C macro definitions

Chip-specific support

- Ready-made C/C++ and assembler peripheral register definition files Multiple code and data models (where applicable)
- Extensive set of language features for PROMable embedded code, including memory keywords, intrinsic functions, interrupt functions, memory-mapped I/O ports, etc.

Linker

- Flexible memory handling allows detailed control of code and data placement

- Removes unneeded functions and variables
- Application-wide type checking of C/C++ variables and functions at link time
- Optional flexible checksum generation for image runtime verification
- Automatic placement of code and data in non-contiguous memory regions

C-SPY Debugger

- Fully integrated debugger for source and disassembly level debugging
- Very fine granularity execution control (function call-level stepping)
- Complex code and data breakpoints
- Versatile monitoring of data
- STL container awareness
- C/C++ call stack window that also shows the function to be entered; double click on any function in call chain updates the editor, Locals, Register, Watch and Disassembly windows to display the state of of that particular function at the time of call
- Trace utility to examine execution history; moving around in the Trace window updates the editor and Disassembly windows to show the appropriate location
- Terminal I/O emulation
- Interrupt and I/O simulation
- C-like macro system to extend debugger functionality
- Application program system calls emulated by the host
- Code Coverage and Profiling performance analysis tools
- Generic flash loader

RTOS support

- OSEK Run Time Interface (ORTI) support included

Libraries and library tools

- All required ISO/ANSI C and C++ libraries and source included
- All low-level routines such as writechar and readchar provided in full source code
- Lightweight runtime library, user-configurable to match the needs of the application; full source included

- Library tools for creating and maintaining library projects, libraries and library modules
- Listings of entry points and symbolic information

Language and standards

- The C programming language as standardized by ISO/ANSI C94 with selected features from C99
- Embedded C++ extended with templates, namespaces, virtual and multiple inheritance and other C++ features that do not cause an overhead in size or speed.
- Full Embedded C++ library containing string, streams etc., as well as the Standard Template Library (STL)
- IEEE-754 floating-point arithmetic
- MISRA C checker for code quality control
- Supports a wide range of industry-standard debug and image formats, compatible with most popular debuggers and emulators, including ELF/DWARF where applicable

User assistance

- Ready-made sample projects and project templates
- Context-sensitive online help with library function lookup
- User guides in PDF format with extensive step-by-step tutorials and reference information
- User-friendly, detailed, and precise error messages and warnings

System requirements

To install and run the IAR Embedded Workbench, you need the following:

- A Pentium-compatible PC with Microsoft Windows XP(SP3), Vista(SP2), or Windows 7. Both 32-bit and 64-bit variants of Windows are supported.
- Internet Explorer 7 or higher
- At least 1 Gbyte of RAM, and 2 Gbytes of free disk space.
- Adobe Acrobat Reader to access the product documentation

3.4.3 Keil PK51 Professional Developer's Kit.

The **Keil PK51 Professional Developer's Kit** is a complete software development environment for classic and extended 8051 microcontrollers. It

includes the tools you need to create, translate, and debug C and Assembly source files. Keil PK51 is easy to learn and use, yet powerful enough for the most demanding 8051 applications. The integrated Device Database configures the tools options for each specific microcontroller

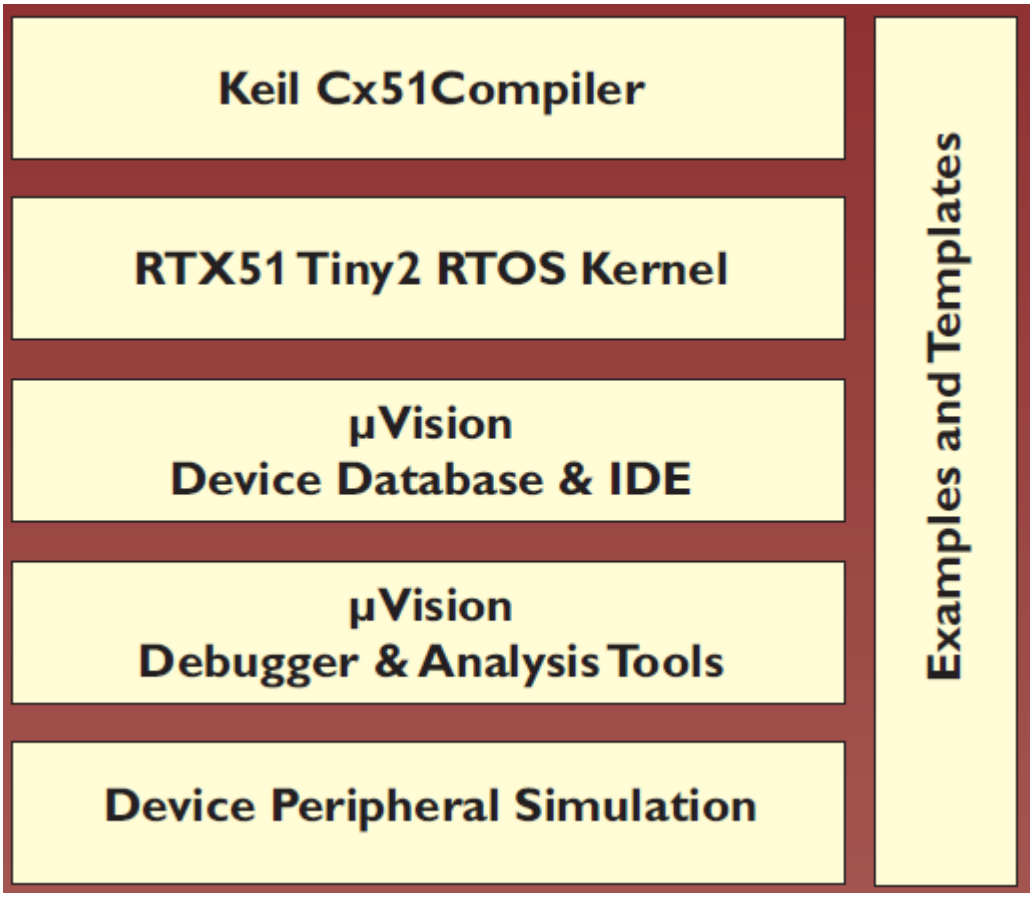


Figure 3.24 KEIL PK51 Professional Developer's Kit

The Keil Cx51 ANSI C Complier supports all classic and extended 8051 device variants. Compiler extensions provide full access to all CPU resources and support up to 16MB memory. Keil Cx51 generates code with the efficiency and speed of hand-optimized assembly. New compiler and linker optimizations shrink programs into the
smallest single-chip devices.

The highlight of Cx51 Complier are:

- Support for all 8051 derivatives and variants
- Fast 32-bit IEEE floating-point math
- Efficient interrupt code and direct register bank control
- Bit-addressable objects
- Sophisticated syntax checking and detailed warnings

- Use of AJMP and ACALL instructions
- Memory banking for code and variables beyond 64KB
- Register parameters and dynamic register variables
- Global program-wide register optimization
- Common code block sub-routine optimization
- Use of multiple data pointers
- Use of on-chip arithmetic units
- Generic and memory-specific pointers
- Re-entrant functions and register bank independent code
- Extensive debug and source browse information
- Simple assembly language interface

RTX51 Tiny2 Real-Time Kernel

The RTX51 Tiny2 multi-tasking real-time kernel makes implementing complex, time-critical software projects easy. RTX51 Tiny2 is royalty-free and is fully integrated into the Keil Cx51 tool chain. It works on all classic 8051 device variants, and supports multiple DPTR and arithmetic units. RTX51 Tiny2 is the successor of the popular RTX51 operating system and provides:

- Single chip and code banking support
- Round robin and cooperative task switching
- Task management with create and delete
- Timeout, Signal, and Ready events for task switching
- Interrupt support for sending signals to tasks

µVision Debugger

The µVision Debugger provides source-level debugging and includes traditional features like simple and complex breakpoints, watch windows, and execution control as well as sophisticated features like performance analyzer, code coverage, and logic analyzer. The µVision Debugger may be configured as a Simulator where programs run on your PC; or as Target Debugger where programs run on your target hardware.

The cycle-accurate µVision Simulator is a software-only product that simulates most features of your 8051/251 device without actually having target hardware. µVision

simulates a wide range of peripherals including I/O Ports, CAN, I²C, SPI, UART, A/D and D/A converter, E²PROM, and interrupt controller. The simulated peripherals depend on the device selected from the µVision Device Database.

Benefits of µVision Device Simulation

- Simulation allows software testing on your desktop with no hardware environment
- Early software debugging on a functional basis improves overall software reliability
- Simulation allows breakpoints that are not possible with hardware debuggers
- Simulation allows for optimal input signals (hardware debuggers add extra noise)
- Signal functions are easily programmed to reproduce complex, real-world input signals
- Single-stepping through signal processing algorithms is possible. External signals stop when the CPU halts
- It is easy to test failure scenarios that would destroy real hardware peripherals

The Keil µVision IDE fully integrates Cx51 Version 9 and provides control of the Compiler, Assembler, Real-Time OS, Project Manager, and Debugger in a single, intelligent environment. With support for all 8051 devices and full compatibility with emulators and third-party tools, Keil Cx51 is clearly the best choice for your 8051 project.

3.4.4 AVR Studio from Atmel

AVR Studio is an Integrated Development Environment (IDE) for writing and debugging AVR applications in Windows 9x/ME/NT/2000/XP/VISTA/7 x32/64 environments. AVR Studio provides a project management tool, source file editor, simulator, assembler and front-end for C/C++, programming, emulation and on-chip debugging.

AVR Studio supports the complete range of ATMEL AVR tools and each release will always contain the latest updates for both the tools and support of new AVR devices.

AVR Studio 4 has a modular architecture which allows even more interaction with 3rd party software vendors. GUI plug-ins and other modules can be written and hooked to the system.

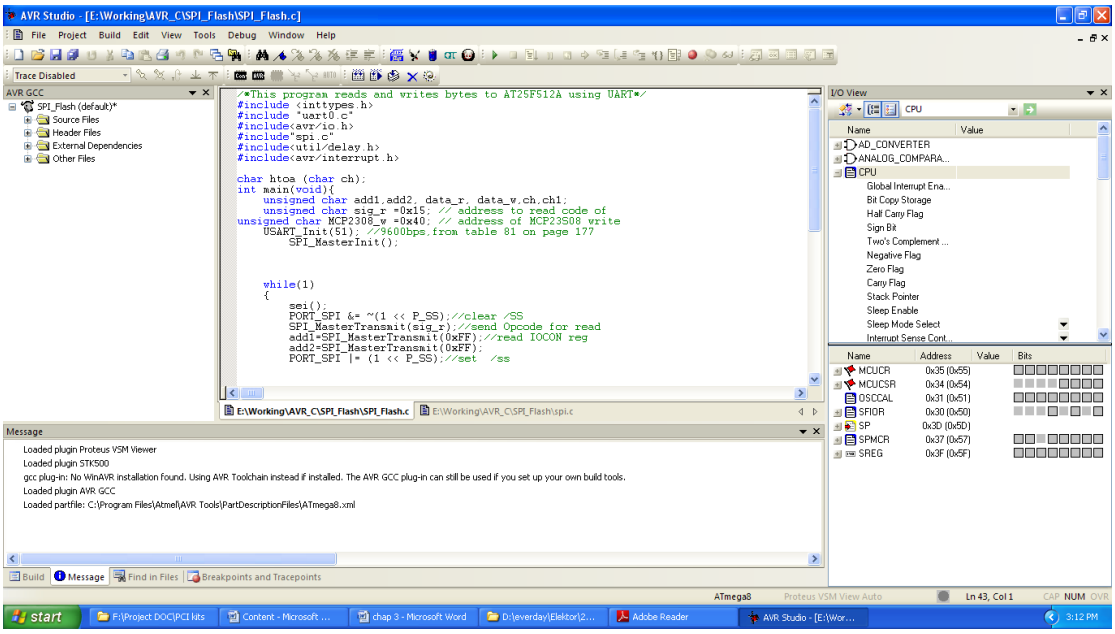


Figure 3.25 AVR Studio IDE

For the present research purpose we are using AVR Studio as the IDE for editing, programming and debugging purpose. More information about the AVR Studio can be found on the CD-ROM provided with the thesis or at www.atmel.com

The Appendix will show the list of AVR devices supported by the AVR Studio. The current version of AVR Studio is AVR Studio 4.18 Service Pack 2 (02/2010)

3.5 Protocols for Programming and debugging.

The set of hardware and software rules followed by both the target and master embedded systems for accomplishing programming and/or debugging of target microcontroller is called protocols for programming and debugging.

The protocols for hardware include the electrical connections between the master and slave microcontrollers. The protocols for software include the transfer of data bytes with help of proper algorithm between the master and target microcontroller

Currently the following protocols are supported by the AVR family of microcontrollers:

Programming Interfaces

- In System Programming (ISP)
- High Voltage Serial Programming (HVSP)
- Parallel Programming (PP)
- JTAG Programming (JTAG Prog)
- PDI Programming on selected devices (PDI)

Debuging Interfaces

- JTAG (JTAG)
- debugWIRE (dW)
- PDI on selected devices (PDI)

The above protocols will be described briefly in the following paragraphs.

In System Programming (ISP)

In-System Programming uses the AVR internal SPI (Serial Peripheral Interface) to download code into the flash and EEPROM memory of the AVR. ISP programming requires only VCC, GND, RESET and 3 signal lines for programming. All AVR devices except AT90C8534, Attiny11 and ATtiny28 can be ISP programmed. The AVR can be programmed at the normal operating voltage, normally 2.7V-6.0V. No high voltage signals are required. The ISP programmer can program both the internal flash and EEPROM. It also programs fuse bits for selecting clock options, startup time and internal Brown Out Detector (BOD) for most devices.

The ISP frequency (SCK) must be less than 1/4 of the target clock. However this requires a 50/50 dutycycle on both target clock and ISP clock. Running a ISP frequency at 1/5 or less than target clock is recommended.

High-Voltage Serial Programming (HVSP)

For High-Voltage programming a 12V programming voltage is applied to the RESET pin of the AVR device. All AVR devices can be programmed with High-Voltage programming, and the target device can be programmed while it is mounted in its socket.

Two different methods are used for High-Voltage programming: 8-pin parts use a serial programming interface, while other parts use a parallel

programming interface. The programming signals are routed to the correct pins of the target device using the cables supplied with STK500.

Parallel Programming (PP)

High pin count AVR devices support the full Parallel Programming (PP) interface. This interface offer high speed programming, and also support programming all fuse and lock bits in the AVR Device. Extreme care should be taken if using PP mode to program an AVR device on an external target. The PP lines do not have level converters, so it is important that the target board is designed to handle 12V on this line. All the programming features are available which are not available while using other programming techniques. If SPI and JTAG programming modes are disabled using PP mode, then it can only be enabled using PP mode only, not any other technique. For using this interface 16 wires are required.

JTAG Programming (JTAG Prog).

AVR devices with JTAG[22] interface also support programming through this interface. The connection for JTAG programming is the same as the JTAG debug interface. A minimum of 6 wires is required to connect JTAG ICE to the target board. These Signals are TCK, TDO, TDI, TMS, VTref and GND.

The programming is done through JTAG TAP controller inside the microcontroller. JTAG specific instruction for AVR is used to program it. Here the JTAG interface is IEEE std. 1149.1 compliant JTAG interface.

The JTAG programming capability supports:

• Flash programming and verifying

• EEPROM programming and verifying

• Fuse programming and verifying

• Lock bit programming and verifying

The present research work will be involving this JTAG interface for programming and debugging the target AVR.

PDI Programming (PDI)

The Program and Debug Interface (PDI) is an Atmel proprietary interface for external programming and on-chip debugging of a device. PDI Physical is a 2-pin interface providing a bi-directional half-duplex synchronous communication with the target device. This interface is only supported on Xmega devices in the A4 series and newer. The PDI supports high-speed

programming of all Non-Volatile Memory (NVM) spaces; Flash, EEPOM, Fuses, Lockbits and the User Signature Row. This is done by accessing the NVM Controller, and executing NVM Controller commands as described in Memory Programming.

Programming Features:

– Flexible communication protocol

– 8 Flexible instructions.

– Minimal protocol overhead.

– Fast 10 MHz programming clock at 1.8V VCC

– Reliable Built in error detection and handling

JTAG debugging.

The JTAG interface consists of a 4-wire Test Access Port (TAP) controller that is compliant with the IEEE 1149.1 standard. The IEEE standard was developed to provide an industry-standard way to efficiently test circuit board connectivity (Boundary Scan). Atmel® AVR devices have extended this functionality to include full Programming and On-Chip Debugging support.

The On-Chip debug system in the AVR is used using JTAG interface. The AVR JTAG ICE from Atmel is a powerful development tool for On-chip Debugging of all

AVR 8-bit RISC Microcontrollers with IEEE 1149.1 compliant JTAG interface. The JTAG ICE and the AVR Studio user interface give the user complete control of the internal resources of the microcontroller, helping to reduce development time by making debugging easier. The JTAG ICE performs real-time emulation of the microcontroller while it is running in a target system.

On-chip Debugging consists mainly of:

• A scan chain on the interface between the internal AVR CPU and the internal peripheral units

• Break Point unit

• Communication interface between the CPU and JTAG system

debugWIRE

The debugWIRE interface was developed by Atmel for use on low pin-count devices. Unlike the JTAG interface which uses 4 pins, debugWIRE makes use of just a single pin (RESET) for bi-directional half-duplex asynchronous communication with the debugger tool. The debugWIRE On-chip debug

system uses a One-wire, bi-directional interface to control the program flow, execute AVR instructions in the CPU.

The debugWIRE interface cannot be used as a programming interface. When the debugWIRE enable (DWEN) fuse is programmed and lock-bits are un-programmed, the debugWIRE system within the target device is activated. The RESET pin is configured as a wire-AND (open-drain) bi-directional I/O pin with pull-up enabled and becomes the communication gateway between target and debugger.

AVR devices with debugWIRE interface are shipped with the DWEN fuse un-programmed from the factory. The debugWIRE interface itself cannot enable this fuse. The DWEN fuse can be programmed through ISP mode, which requires connection to a 6-pin header. For this reason it is recommended to place the full 6-pin ISP connector on your target board to simplify emulation and programming.

PDI debugging

The Program and Debug Interface (PDI) is an Atmel proprietary interface for external programming and on-chip debugging of the device. The PDI supports high-speed programming of all Non-Volatile Memory (NVM) spaces; Flash, EEPOM, Fuses, Lockbits and the User Signature Row. This is done by accessing the NVM Controller,

and executing NVM Controller commands as described in Memory Programming.

The On-Chip Debug (OCD) system supports fully intrusive operation. During debugging no software or hardware resources in the device is used (except for four I/O pins required if JTAG connection is used). The OCD system has full program flow control, supports unlimited number of program and data breakpoints and has full access (read/write) to all memories.

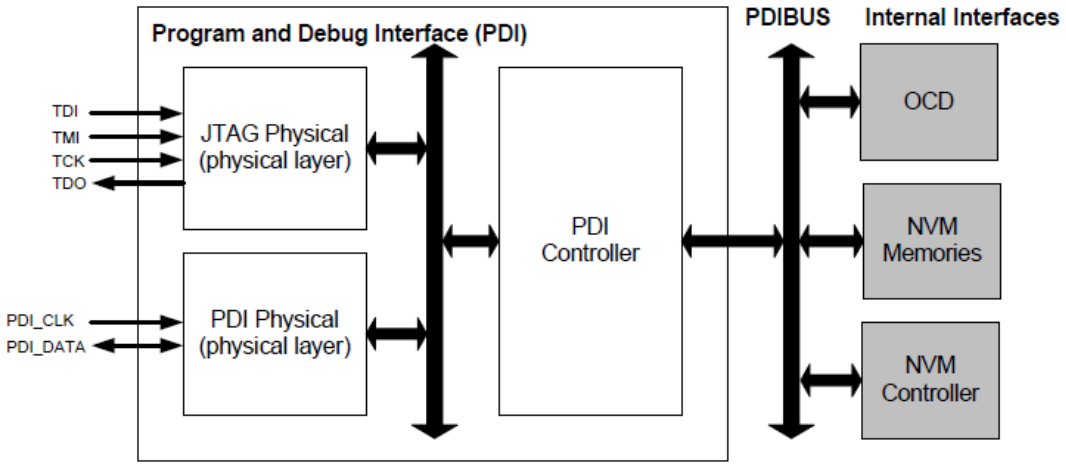


Figure 3.26 The PDI with JTAG and PDI physical and closely related modules (grey)

Both programming and debugging can be done through two physical interfaces. The primary interface is the PDI Physical. This is a 2-pin interface using the Reset pin for the clock input (PDI_CLK), and the dedicated Test pin for data input and output (PDI_DATA).

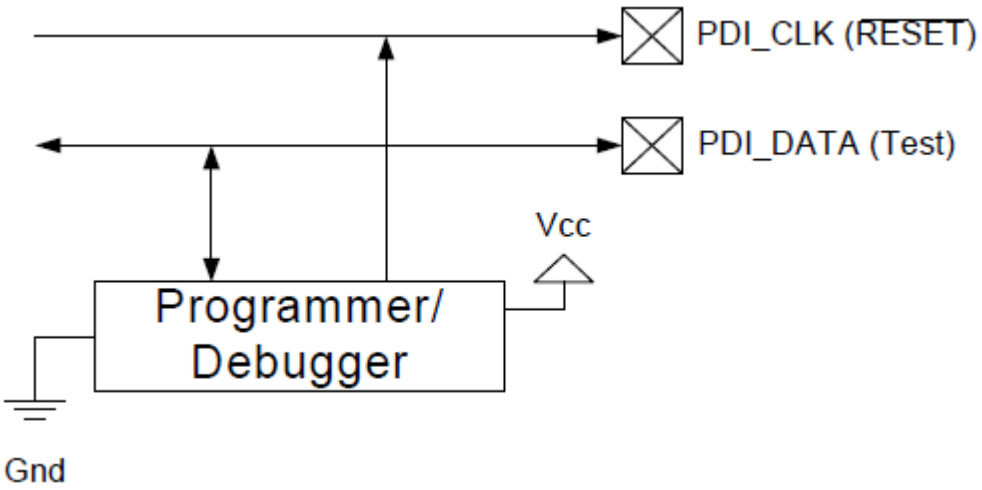


Figure 3.27 PDI Connections

Debugging Features:

– Non-Intrusive Operation

- Uses no hardware or software resource

– Complete Program Flow Control

- Symbolic Debugging Support in Hardware
- Go, Stop, Reset, Step into, Step over, Step out, Run-to-Cursor

– 1 dedicated program address breakpoint or symbolic breakpoint for AVR studio/emulator

– 4 Hardware Breakpoints

– Unlimited Number of User Program Breakpoints

– Uses CPU for Accessing I/O, Data, and Program

– High Speed Operation

- No limitation on system clock frequency

3.6 Programmers and debuggers.

Different programmers and debuggers available in the market to be used with different popular microcontrollers are listed in the table 3.2 below:

Microcontroller Family	Company/Make	Name of programmer Debugger	Protocol
ARM7,9,11, Cortex-M0, M1, M3, M4, R4, Renesas RX	Segger.com	J-Link JTAG/SWD Emulator with USB interface	JTAG/SWD
PIC, dsPIC	Microchip	MPLAB® PM3 Universal Device Programmer	Integrated In Circuit Serial Programming™ (ICSP™)
8051 Family	Reynolds Elec.	Flash LAB - 51	Serial
AVR	Atmel	AVR Dragon	ISP,JTAG, PDI,dW,HVPP,HVSP
AVR	Atmel	AT STK500	HVPP,HVSP,ISP
AVR	Atmel	AVR ONE	HVPP,HVSP,ISP

MSP430	TI	MSP-FET430 Flash Emulation Tool	2-wire/4-wire JTAG

3.7 Journals/Paper Reviews.

The references of articles and research papers from various magazines, journals and company white papers gives complete information and data for any research work.

The basic motivation was taken from US patent 20080126655 "**SINGLE PCI CARD IMPLEMENTATION OF DEVELOPMENT SYSTEM CONTROLLER, LAB INSTRUMENT CONTROLLER, AND JTAG DEBUGGER**". This patent describes a system for integrating multiple electrical system testing functions into a single peripheral component interconnect (PCI) card. The functions of system bring-up and debug are integrated into a single PCI card, which utilizes an operating system and a set of industry standard interfaces to interconnect with standard lab instrumentation. The integrated PCI card utilizes an embedded high performance microprocessor and a compact operating system to provide control over: system-under-test (SUT) power on/off; system device sequencing via programmable General Purpose Input/Output (GPIO); system parametric control (e.g. voltage, temperature, and frequency); system parametric measurement; system debug; and remote control operation via internet interface. In one embodiment, the integrated PCI card comprises the instrumentation controller, Joint Test Action Group (JTAG) Debugger, SUT system controller, and a computer-controlled GPIO card in a single, self aware, half-slot PCI card.

Reconfigurable PCI Bus interface(RPCI) from Central Laboratory for Electronics, Germany uses Xilinx 4010E chip package 208 with speed grade 3C, developing their own core for PCI interface. The idea behind the making of PCI based embedded training system came from the Development Kit, Cyclone™ II Edition which was worked out at one of the Altera exhibition.

There were also few papers that led to development of this research work.

A FPGA-based PCI bus interface for real-time log-polar image processing system from Dpto. Inform´atica y Electr´onica - Universitat de Val`encia, Spain, describes about the development of a PCI bus interface for

a log-polar image acquisition system. The characteristics of the log-polar camera employed in the system, and its interface to a real-time image processing system, bind an ad-hoc solution that is difficult o achieve with any general purpose image acquisition system. The main objective was to develop a card allowing log-polar camera image acquisition, camera and image processing system intercommunication, and PC user and other systems interface. The card has a PCI bus interface designed using a complex programmable logic device that holds the interface and other logic control for the system in the same chip.

An FPGA-Based Coprocessor for ATM Firewalls appeared in the Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'97), Napa, CA, April, 1997. An agile firewall coprocessor is described that is based on field programmable gate array (FPGA) technology. This implementation of the firewall enables a high degree of traffic selectability yet avoids the usual performance penalty associated with IP level firewalls. This approach is applicable to high-speed broadband networks, and Asynchronous Transfer Mode (ATM) networks are addressed in particular. Security management is achieved through a new technique of active connection management with authentication. Past approaches to network security involve firewalls providing selection based on packet filtering and application level proxy gateways. IP level firewalling was sufficient for traditional networks but causes severe performance degradation in high speed broadband environments. The approach described in this paper discusses the use of an FPGA-based front end processor that filters relevant signaling information to the firewall host while at the same time allowing friendly connections to proceed at line speed with no performance degradation.

PCI Bus Interface Card for Communication at 2 Mbps from 1Departamento de Engenharia Electrotécnica, Pólo II - Pinhal de Marrocos 3030 Coimbra. A solution that implements the interface between a Personal Computer (PC) and a rented Integrated Services Digital Network (ISDN) E1 line, is presented. The card is fully compliant with PCI (Peripheral Component Interface) technology, and allows full-duplex communication through standard serial

lines at 2 Mbps. The solution is also suitable to other applications that require some processing of the bitstream, such as speech recognition, PBX, etc.

Overview of the use of the PCI bus in present and future high energy physics data acquisition systems published in the PCI'95 Conference Proceedings March 1995 pg. 83-88. Due to its very complex data acquisition systems High Energy Physics (HEP) experiments are always looking for cheap and fast computers and communication equipment. PCI as a mainstream product is one of the new technologies responding to these criteria. After a short introduction of CERN and its

Particle Physics Facilities, the first part of this article describes, with a real development project as example, the specific problems of data acquisition systems in HEP experiments at the future LHC accelerator. Solutions where PCI technology will play a role will be presented, showing as examples the use of a VMEbus module with dual port ram and PCI to SCI interfaces. The second part describes the NA48 experiment including a detailed description of the development of the PCI to HIPPI interface.

US Patent "**UART Emulator Card**" describes a interface device and corresponding method for coupling a peripheral controller to a host computer, the interface device including an emulated universal asynchronous receiver transmitter(UART) for the host computer. The interface device further includes a plurality of registers, preferably a control, status, and data register, such as a multi-register data buffer, corresponding to registers of a UART, a host computer port, preferably compatible with a PCMCIA standard, that includes an address map for the plurality of registers, a peripheral controller port providing an address mapped parallel interface to the plurality of register and control logic for providing status signals, including UART status signals, to the host computer port and to the peripheral controller port. The interface device may further include a pacing circuit for substantially emulating, preferably dependent on a baud rate of the data information, the timing limitations of the UART.

United States Patent 5968156 "**Programmable peripheral component interconnect (PCI) bridge for interfacing a PCI bus and a local bus having reconstructable interface logic circuit therein**" describes a peripheral component interconnect (PCI) bridge which interfaces between PCI

and local buses to provide a communicator for performing a communication between peripheral devices connected to the PCI bus and system devices connected to the local bus, is provided. The PCI bridge comprising a PCI register which is initialized according to a reset signal from the PCI bus and then stores configuration information on the PCI bus, a local register which is initialized according to a reset signal from the PCI bus and then stores configuration information on the local bus, PCI bus interface logic for performing interfacing according to the configuration information stored in the PCI register, local bus interface logic for performing interfacing according to the configuration information stored in the local register, and a logic transformer for reconstructing the PCI bus interface logic according to a command input from a user.

United States Patent 5737524 "**Add-in board with programmable configuration registers for use in PCI bus computers**" describes an adapter or add-in card for using in a peripheral component interconnect (PCI) computer includes a universal module which couples the card to the PCI bus. The module includes a set of selectively programmable configuration registers which are loaded by a microprocessor on the adapter. A circuit arrangement on the module issues a command which inhibits the PCI processor from accessing the configuration registers until fully loaded.

3.8 Outcome of Literature Survey.

This chapter 3 includes everything about literature survey that defends the research work for developed PCI card. Looking at the future market demands learning to develop PCI cards is also required, even though easy PnP interfaces like USB are extensively used.

Actually US patent 20080126655 is a highly complex and sophisticated PCI system, so as an approach to such highly complex system it was thought to develop a minimal PCI card that have function of only debugging and programming target microcontroller, in short use as PCI based microcontroller trainer system. To put in process such a system different PCI cards, debuggers and programmers and microcontroller families were reviewed and on the basis of requirement specification and analysis done in chapter 5 the AVR microcontroller is selected as target microcontroller and MCS9835CV is selected as PCI controller chip.

Chapter 4

Theoretical background

Chapter 4 Theoretical Background

The theoretical aspect of the devices and technologies used in the research work is to be discussed before proceeding towards designing of the PCI bus card. This chapter gives detail description of the following major topics that are to be concentrated on.

- Peripheral Component Interconnect Bus.
- AVR 8 – bit Microcontroller family.
- PCI Interfacing chip.
- Microcontroller Programming Protocols – SPI and JTAG.

4.1 Peripheral Component Interconnect Bus.

4.1.1 About the PCI-SIG group and evolution of PCI bus

Formed in 1992, the PCI-SIG is the industry organization chartered to develop and manage the PCI standard. With over 900 members, the PCI-SIG effectively places ownership and management of the PCI specifications in the hands of the developer community. A Board of Directors comprised of nine people, each elected by the membership, leads the PCI-SIG.

The PCI-SIG is chartered to:

- Maintain the forward compatibility of all PCI revisions or addenda
- Contribute to both the establishment of PCI as an industry-wide standard and to the technical longevity of the PCI architecture
- Maintain the PCI specification as a simple, easy-to-implement, stable technology that supports the spirit of its design

The PCI-SIG fulfills its charter by continuing to promote innovation and evolve the PCI standard to meet the industry's needs. Through interoperability testing, technical support, seminars and industry events, the PCI-SIG enables its members to generate competitive and quality products.

The figure 4.1 shows the evolution of PCI Bus family history.

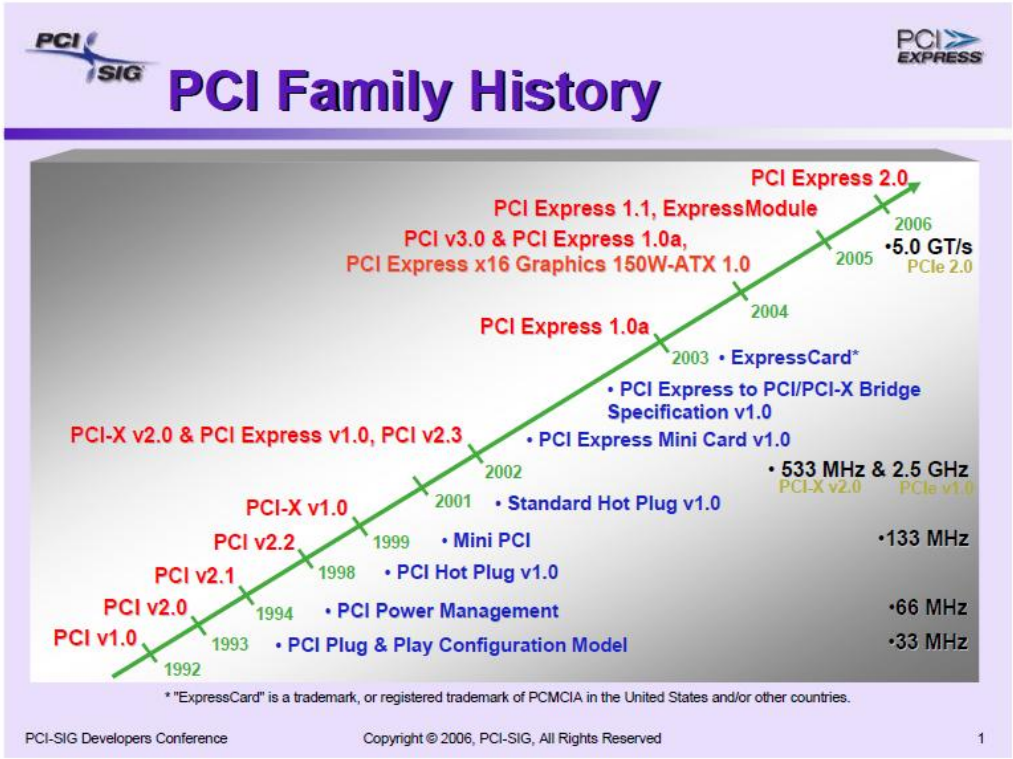


Figure 4.1 PCI Family History

4.1.2 Introduction to the PCI Bus Architecture

The PCI (Peripheral Component Interconnect) local bus [3] is a high speed bus. The PCI Bus was proposed at an Intel Technical Forum in December 1991, and the first version of the specification was released in June 1992. The current specification of the PCI bus is revision 2.1, which was released on June 1, 1995. Revision 2.2, which is due to be published soon, would add support for Hot insertion and removal of cards, power management, and incorporate several other ECRs that have been accumulated since Revision 2.1.

Since its introduction, the PCI bus has gained wide support from all the computer industry. Almost all PC systems today contain PCI slots, as well as the Apple and IBM Power-PC based machines, and the Digital Alpha based machines. The PCI standard has become so popular it influenced the creation of more than one related standard based on leveraging PCI technology. The PCI Bus is designed to overcome many of the limitations in previous buses. The introduction of PCI architecture was discussed in chapter 3 3.11.

The major benefits of using PCI are:

- High speed

The PCI bus can transfer data at a peak rate of 132MBytes/sec using the current 32 bit data path and a 33MHz clock speed. Future implementations featuring a 64 bit data path and 66MHz clock speed may transfer data as fast as 524MBytes/sec. Even at its basic mode, PCI delivers more than tenfold the performance levels offered by its predecessor in the PC world, the ISA bus.

- Expandability

The PCI bus can be expanded to a large number of slots using PCI to PCI bridges. The bridge units connect separate small PCI buses to form a single, unified, hierarchical bus. When traffic is local to each bus, more than one bus may be active concurrently. This allows load balancing, while still allowing any PCI Master on any bus to access any PCI Target on any other PCI bus.

- Low Power

Motherboards can lower their power requirement by reducing the clock rate as low as 0Hz (DC). All PCI compliant cards are required to operate in all frequency ranges from 0Hz to 33MHz.

- Automatic Configuration

All PCI compliant cards are automatically configured. There is no need to set up jumpers to set the card's I/O address, IRQ number, or DMA channel number. The PCI BIOS software is responsible for probing all the PCI cards in a system, and assigning resources to every card, as required.

- Future expansion

The PCI specification can support future systems by incorporating features such as an optional 64 bit address space, and 66MHz bus speed. The specification defines enough reserved fields in all the bus definitions (configuration space registers, bus commands, and addressing modes), that any unforeseen enhancement will not hinder compatibility with present systems.

- Portability

By incorporating the (optional) OpenBoot standard, any device with OpenBoot Firmware can boot systems containing any microprocessor and O/S. Even without OpenBoot, it is common to see drivers for many video cards and SCSI controller for multiple CPU architectures and operating systems.

- Complex memory hierarchy support

The PCI Bus supports advanced features such as bus snooping to allow cache coherency to be kept even with multiple bus masters, and a locking mechanism to support semaphores.

- Interoperability with existing standards

The PCI Bus allows interoperability with existing ISA cards by supporting subtractive decoding of addresses (allowing addresses not decoded by PCI cards to be routed to an ISA backplane). The standard also supports the fixed legacy addresses for VGA cards and IDE controllers (required for system boot). Another feature supporting backward compatibility allows different devices to respond to different I/O byte addresses even if they share the same 32 bit word.

The PCI device can be identified by its functions and device numbering as shown in figure 4.2

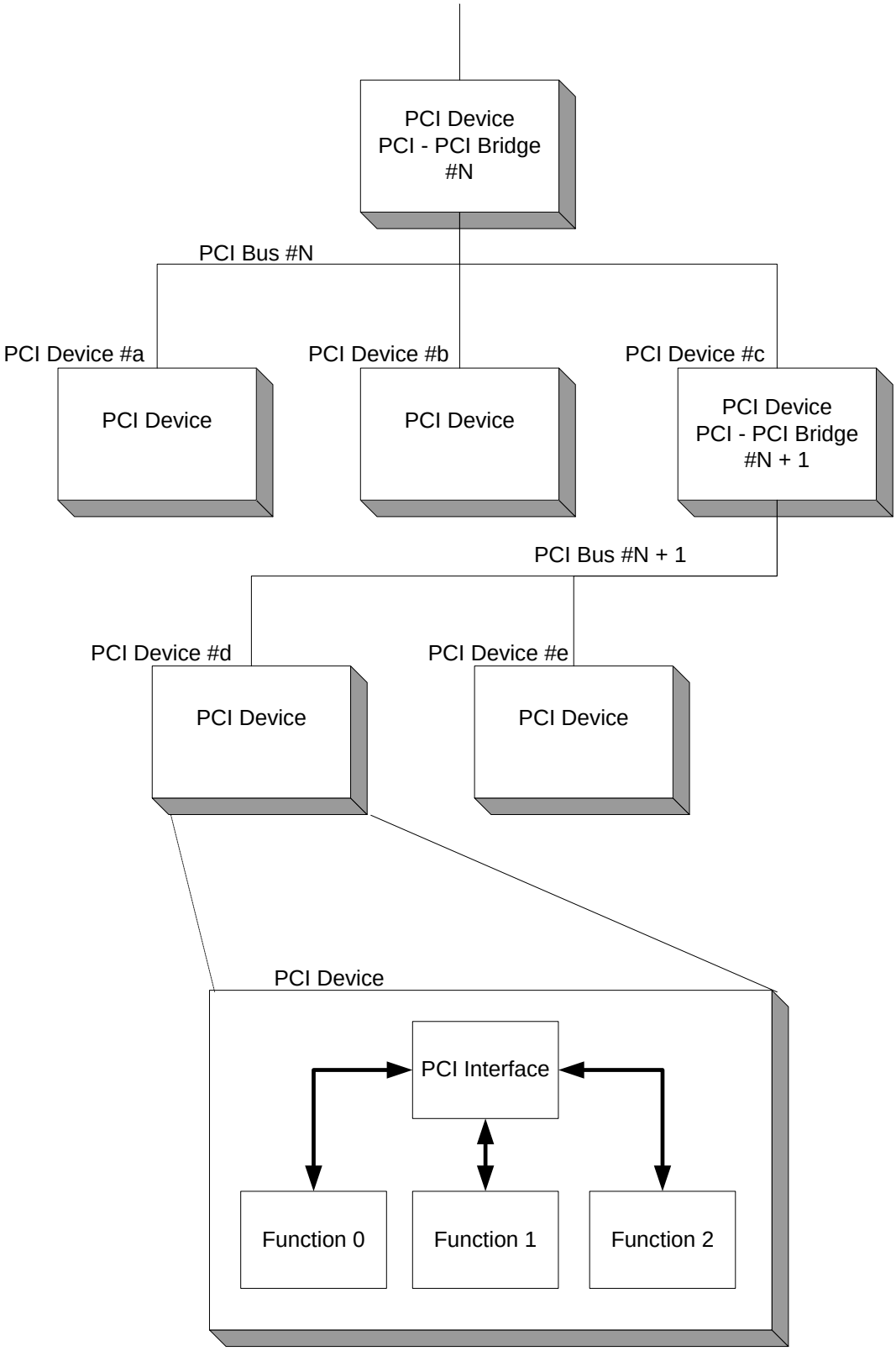


Figure 4.2 PCI Device numbering scheme

Each single device can have eight internal functions numbering from 0 to 7. The device is separate hardware entity. These functions of same device share

the same PCI bus. A specific function is selected by ‘Bus number/Device number/Function number’

The PCI (Peripheral Component Interconnect) local bus is a high-performance connection between motherboard components and expansion boards. It was first proposed at an Intel Technical Forum in December 1991, and the first version of the specification was released in June 1992. The PCI Steering Committee members who developed the initial version of the specification were Compaq, DEC, IBM, Intel, and NCR. In June 1992, PCI became an open industry standard controlled by the newly formed PCI SIG (PCI Special Interest Group). In April 1993, revision 2.0 of the PCI specification was released.

Vendor support for the PCI standard has been widespread. Apple has announced that it will support the PCI bus in a future version of its PowerPC RISC Macintosh. (Note that first-generation PowerPC Macs will use the NuBus to support existing NuBus peripheral boards.) DEC intends to support PCI in its Alpha-based systems, and the company's high-speed DECchip 210066 RISC processor implements a PCI interface on the chip itself. This kind of support, in addition to that from many vendors of 80x86-based systems, paves the way for PCI to become a universally accepted component interconnect and expansion-bus standard.

The motivation for PCI was the fact that PC I/O architecture was so slow in relation to the processor that further improvements in processor technology would not have produced any noticeable improvements in overall system performance. PCI removes systems designers from the processor treadmill by isolating the I/O subsystem from the processor/memory/cache subsystem.

A fundamental part of a PCI design is the PCI-to-host bridge chip that connects the PCI bus to the processor's local bus. PCI peripherals are then connected directly to the PCI bus. Once a host bridge chip is available, a new processor has access to all available PCI components. This allows the PCI bus standard to be processor independent.

When a new processor becomes available, only the PCI-to-host bridge chip needs to be replaced; the rest of the system remains unchanged. PCI is a component- and board-level bus; other I/O buses, such as a SCSI bus, can be included in a PCI system with a controller chip or a board that interfaces to

the PCI bus. Designers of I/O components such as graphics, SCSI, or LAN controllers can now concentrate on improving the performance of their products instead of continually redesigning their products for different processor speeds and bus types.

Although processors are quickly moving past 33-MHz operation, the PCI bus is defined to operate only up to 33 MHz. The processor speed can be faster than the PCI bus speed, however, because the host-to-PCI bridge isolates the processor bus from the PCI bus. The bridge chip can contain buffering to enhance performance and bus utilization. This is especially useful when using a processor, such as the 486, that doesn't support burst writes. Since the PCI bus does support burst writes, the host bridge can buffer a nonburst write from the processor and present it to the PCI bus as a burst write. A bridge might contain other system-support logic, such as for a cache controller or for handling the PCI bus's centralized arbitration mechanism.

PCI won't replace the standard expansion buses that are popular today, but instead it will complement them. These expansion buses are added to a PCI system by the inclusion of a PCI-to-expansion bus bridge chip. The standard expansion buses (e.g., ISA, EISA, Micro Channel, and NuBus) are lower in performance, but they provide for maximum system expandability for less-demanding peripherals.

4.1.3 PCI Bus Design

The design goals of the PCI bus standard were threefold. First, it would produce a low-cost, high-performance local-bus interface. Second, it would provide automatic configuration of components and add-in boards. Finally, the design would have the versatility to support future generations of peripherals.

The cost and size of connectors, chips, and motherboard traces increase as the number of pins needed to implement a bus increases. The PCI standard reduces costs by using a multiplexed address and data bus that reduces the pin count and size of the components. A PCI target (which is defined later) can be implemented with only 47 pins, and a PCI master can be implemented with only 49 pins.

Despite the small number of lines, this setup can manage bus addressing, data transfer, arbitration logic, and interface control. The figure "PCI Interface Signals" lists the PCI bus pins, with both the required and optional pins

shown. Note that the optional pins provide support for a cache and atomic operations. Other optional pins implement a 64-bit bus, giving the PCI standard a growth path for future systems.

Two sizes of add-in boards are specified: a standard length (about 12 inches long) and a short length (about 7 inches long). Note that the PCI bus is limited to 10 loads. Chips on the motherboard take up one load each, and add-in boards in PCI slots consume two loads each. For example, if devices on the motherboard consume four loads, this leaves six loads available, allowing three PCI slots. Too much loading can cause signals to violate timing specifications, which leads to system failures.

PCI Universal Card 32/64 bit

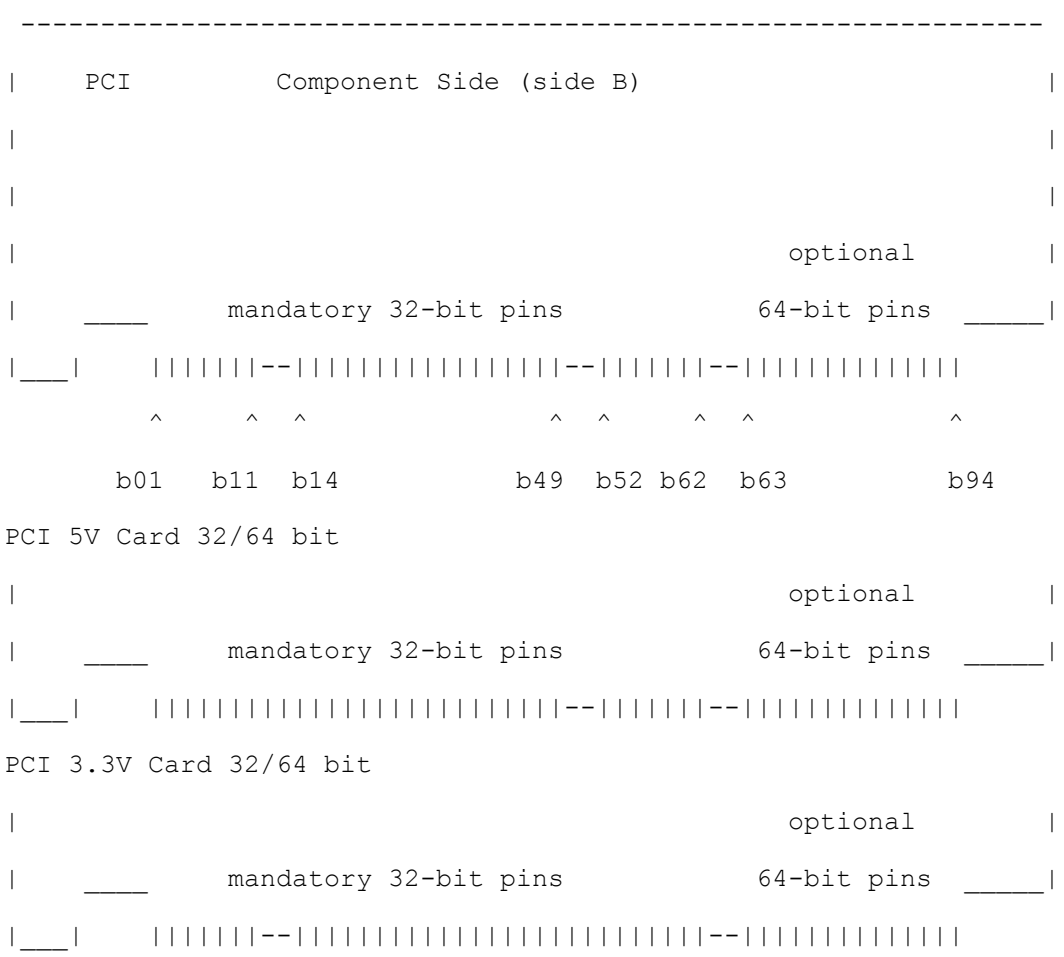


Figure 4.3 Universal PCI Slot

The PCI specification defines two types of connectors that may be implemented at the system board level: One for systems that implement 5 Volt signaling levels, and one for systems that implement 3.3 Volt signaling

levels. In addition, PCI systems may implement either the 32-bit or 64-bit connector. Most PCI buses implement only the 32-bit portion of the connector which consists of pins 1 through 62. Advanced systems which support 64-bit data transfers implement the full PCI bus connector which consists of pins 1 through 94. Three types of add-in boards may be implemented: "5 Volt add-in boards" include a key notch in pin positions 50 and 51 to allow them to be plugged only into 5 Volt system connectors. "3.3 Volt add-in boards" include a key notch in pin positions 12 and 13 to allow them to be plugged only into 3.3 Volt system connectors. "Universal add-in boards" include both key notches to allow them to be plugged into either 5 Volt or 3.3 Volt system connectors.

To accommodate the industry's shift from 5-V signals to 3.3-V signals, PCI also defines a 3.3-V PCI specification. PCI add-in boards and slots are keyed so that 5-V boards plug only into corresponding 5-V PCI slots and 3-V boards plug only into 3-V slots. The PCI SIG is encouraging add-in board vendors to design "universal" boards that operate at both voltages and can plug into either a 5- or a 3.3-V PCI slot (see the figure 4.3).

PCI defines three types of address space: memory, I/O, and configuration. Configuration space is a 256-byte area inside each PCI device that contains information about the device. Such information includes the type code, which indicates whether the device controls a mass-storage unit, a network interface, a display, or other hardware. Multifunction devices are supported, as long as there is only one physical connection to the PCI bus, such as one chip that supports both SCSI and Ethernet. The configuration space also contains the PCI control, status, and latency timer registers; the location of the device's expansion ROM; and the base-address registers. At power-up, the system scans the configuration space of all devices on the PCI bus and then assigns each device a unique base address and an interrupt level.

A PCI device's expansion ROM contains code to initialize the device, or devices. The ROM's contents are arranged in little-endian (i.e., Intel) format, but there's nothing in the design to prohibit other processors from making use of the information. The ROM can contain different code images that initialize the device for different processor architectures. PCI provides true plug-and-play; the system takes care of hardware configuration automatically. A user just plugs in a board, and it works.

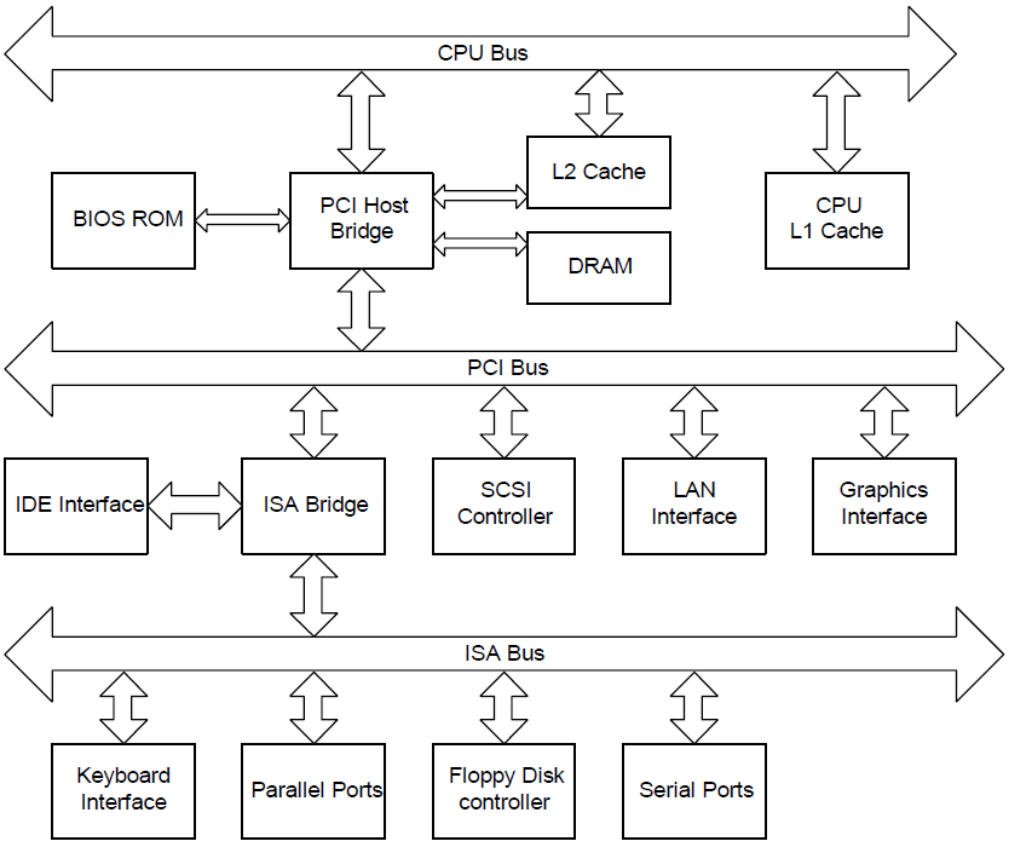


Figure 4.4 PCI Bus System Architecture

4.1.4 PCI Bus Operation

A PCI transaction takes place between a master and a target. Master is a term used by the PCI standard for a bus master (i.e., a device that takes control of the bus and initiates accesses, such as a processor or a bus-mastering SCSI controller). Target is a term for a slave device (i.e., a device that only responds to accesses, such as memory or a VGA controller).

The PCI bus does all transfers as burst reads and writes to optimize performance. A normal bus access requires a new address before each data transfer, whereas a burst transfer starts with one address and then performs multiple data transfers to or from consecutive address locations. For PCI, a burst transfer begins with an address phase, followed by one or more data phases. This allows one data transfer to be performed during each clock period. This gives a 33-MHz PCI bus a burst transfer rate of 132 MBps on a 32-bit bus and 264 MBps on a 64-bit bus. Bursts can be either to the memory or to the I/O space.

PCI burst transfers have an indefinite length, whereas most other architectures have a limited, fixed burst length. PCI bursts continue until the master or target requests the transfer to end or until a higher-priority device needs to use the bus. Each PCI device has a latency timer that defines the longest period of time that that device is allowed to use the PCI bus. This timer is accessed through the configuration space and is programmed by the processor. The processor can optimize overall system performance by intelligently programming the latency timers of all devices on the PCI bus.

The master specifies the type of burst transaction by using bus commands. Another way to improve the performance of any bus running at a particular speed is to reduce its overhead, which is time that the bus is in use but data is not being transferred. When a PCI device wants to access the PCI bus, it must request use of the bus from the central arbiter (which can be located in the PCI-to-host chip) by using its REQ (request) signal. The arbiter uses its GNT (grant) signal to allow the device to use the bus. On most buses in use today, the arbitration process takes up cycles that could be spent transferring data. PCI eliminates this loss by allowing the arbitration for the next access to occur while the current access is still in progress.

The PCI bus can transfer data 32 bits or 64 bits at a time. The optional signals REQ64 and ACK64 allow a 64-bit master to ask if a target is capable of 64-bit data transfers. A 64-bit transfer occurs only if a master asks for a 64-bit transfer and a target responds that it can do such transfers.

A PCI master can support 32- or 64-bit addressing regardless of whether it supports a 32- or a 64-bit data path. Since a master might not be able to look at a target's configuration space, a master that supports 64-bit addressing doesn't always know if a target supports 32- or 64-bit addressing. Therefore, the master must be able to present an address in a format that either type of target can accept. The master addresses 32- or 64-bit devices capable of 64-bit addressing by issuing a dual-address-cycle command.

To implement such a command, during the first address cycle the master first drives the lower 32 bits of the 64-bit address on the lower 32 bits of the bus, and the upper 32 bits of the 64-bit address on the upper 32 bits of the bus. During the second address cycle, the upper 32 bits of the 64-bit address are driven again, but this time on the lower 32 bits of the bus.

A 64-bit target will take in the full 64 bits of the address on the first clock cycle, ignore the second address phase, and decode the 64-bit address with no additional delay. A 32-bit target must wait an additional clock cycle to receive the full 64-bit address, since it is transferred in two 32-bit portions. Targets that support only 32-bit addressing are mapped into the lower 32 bits (4 GB) of address space, so they can be accessed by masters that support 32- or 64-bit addressing transparently.

4.1.5 The PCI Bus Signal Description

The PCI Bus is a multiplexed, synchronous, 32 bit bus. A minimal PCI target interface consists of 47 signals, while a master or master/target interface requires a minimum of 49 signals. The following paragraphs summarize the PCI signals. It is not intended to be a PCI design reference, only to illustrate very generally how PCI works. In order to design a fully compliant PCI card, please refer to the PCI standard [(PCISIG, 1995)].

4.1.5.1 Signal types

The PCI standard defines a few signal types:

- t/s

  Tri State. Signals of this type are shared bus signals and may be driven by only one driver at a time. The rest of the bus peripherals should tri-state signals of this type when not in use. When one driver stops driving a t/s signal, a new driver must wait at least one cycle before it can drive the same signal. This is called bus turnaround and is used to prevent any case where two or more drivers are trying to drive the same line.

  Example: AD[31:0], /C/BE[3:0], PAR, /REQ, /GNT, AD[63:32], /C/BE[7:4], PAR64.

- s/t/s

  Sustained Tri-State. Signals of this type are tri state signals. The difference between these signals and normal t/s signals is that any driver driving these signals must drive them high for at least one cycle before tri-stating them. By driving the signal high, the line is charged so when it is tri-

stated one cycle later, it will stay high. This allows rapid switching from one driver to another, without undefined logic levels in between.

Example: /FRAME , /TRDY , /IRDY , /STOP ,/LOCK , /DEVSEL , /PERR , /REQ64 , /ACK64 .

- o/d

  Open Drain. Signals of this type are wired-ORed. Multiple drivers may drive this signal to a low state. When no driver is driving the signal, a pull-up resistor is used to keep it in the high state.

  Example: SERR , /INTA , INTB , INTC , INTD .

- in

  Input. Signals of this type are always input.

  Example: CLK, /RST , IDSEL, TCK, TDI, TMS, TRST .

- out

  Output. Signals of this type are always output.

  Example: TDO.

4.1.5.2 System Signals

CLK

All the PCI Bus signals are fully synchronized to the PCI Clock signal, **CLK**.

RST

This is the PCI Bus reset signal. It is asynchronous to the PCI clock signal.

4.1.5.3 Address/Data and Command

AD[31:0] t/s

This is the multiplexed address/data bus. All PCI transactions begins by a Master driving the address on the 1st cycle. When the Master is doing a read transaction, the 2nd cycle is a turnaround cycle in which the Master tri-states the bus and the target enables it's data buffers to drive the results on the bus. The actual data is transferred only beginning on the 3rd cycle onwards. When a Master is doing a write transaction, it can begin driving the write data on the 2nd cycle, because no turnaround cycle is needed (the bus does not change direction).

C/BE [3:0] t/s

These are dual function pins, and are always driven by the Master. During the 1st cycle, when **AD[31:0]** is driving the address, **C/BE [3:0]** is driving the bus command (Memory read, Memory
write, etc.). When **AD[31:0]** is driving the data, **C/BE [3:0]** drive the byte enable for each byte lane on the data bus. During a write transaction, the Master drives **C/BE [3:0]** with information specifying which byte lanes contain valid data. During a read transaction, the Master drives **C/BE [3:0]** with information specifying which byte lanes are requested by the master.
The byte enable signals are active low and are summarized in the following table 4.1:

Byte Enable	**Signal Maps To**
C/BE 3	AD[31:24]
C/BE 2	AD[23:16]
C/BE 1	AD[15:8]
C/BE 0	AD[7:0]

Table 4.1 Byte Enable signals

PCI Byte Enabling Mappings
The PCI Bus commands are:

C/BE 3	C/BE 2	C/BE 1	C/BE 0	Command Type
0	0	0	0	Interrupt Acknowledge
0	0	0	1	Special Cycle
0	0	1	0	I/O Read
0	0	1	1	I/O Write
0	1	0	0	Reserved

0	1	0	1	Reserved
0	1	1	0	Memory Read
0	1	1	1	Memory Write
1	0	0	0	Reserved
1	0	0	1	Reserved
1	0	1	0	Configuration Read
1	0	1	1	Configuration Write
1	1	0	0	Memory Read Multiple
1	1	0	1	Dual Address Cycle
1	1	1	0	Memory Read Line
1	1	1	1	Memory Write and Invalidate

Table 4.2 PCI Commands

PAR t/s

The PAR signal used to ensures even parity across the AD[31:0] and C/BE [3:0] signals, and is always valid one cycle after the information on AD[31:0] is valid, i.e.:

1. One cycle after an address phase (1st PCI cycle).
2. One cycle after a Master asserts IRDY on a write transaction (i.e. data ready to be written).
3. One cycle after a Target asserts TRDY on a read transaction (i.e. data is ready to be read).

4.1.5.4 Interface Control

/FRAME s/t/s

The /FRAME signal is used to start a PCI transaction. When the PCI bus is Idle, /FRAME is high. When a Master begins a new PCI transaction, /FRAME is driven low. /FRAME is kept low during all the transaction, until (but not including) the last data item transferred. When a Master read or writes the last data item in a burst access cycle, /FRAME will be driven high when IRDY is driven low for the last time.

/IRDY s/t/s

/IRDY is used by the Master to indicate its readiness to perform data transfer on a word by word basis. When a Master is reading data, it will drive /IRDY high, until it can accept a data word. The Master will then drive /IRDY low on the same cycle it samples the data from the Target. When a Master is writing data, it will drive /IRDY high until AD[31:0] contains a valid data word to be written. It will then drive /IRDY low on the same cycle AD[31:0] is valid for write. The actual data transfer will take place only when both /IRDY and TRDY are active. Once driven low, /IRDY cannot be driven high until the current bus transaction is done (i.e. /TRDY or STOP also driven low).

TRDY s/t/s

/TRDY is used by the Target to indicate its readiness to perform data transfer on a word by word basis. When a Target is accepting data (Master writing data), it will drive /TRDY high until it can accept the data word. The Target will then drive /TRDY low on the same cycle it samples the data item from AD[31:0]. When a Target is supplying data (Master reading data), it will drive /TRDY high until it can drive AD[31:0] with the requested data. The Target will then drive /TRDY low on the same cycle it drives the requested data item on AD[31:0]. The actual data transfer will take place only when both /IRDY and /TRDY are active. Once driven low, /TRDY cannot be driven high until data is transferred (i.e. IRDY also driven low).

DEVSEL s/t/s

/DEVSEL is used by the Target to acknowledge the Master it is handling the current bus cycle. When a Master begins a read/write transaction, it drives /FRAME low on the 1st cycle, together with the requested address on AD[31:0]. All the Targets on the bus recognize the beginning of a new transaction by /FRAME going low, and compare the address on AD[31:0] with their base address registers. The Master expects one Target, who’s address

range contains the address driven on AD[31:0] to respond with a low DEVSEL within 4 clock cycles. If no target responds within 4 cycles, the Master assumes that no Target exists on the bus at the specified address, and the transaction is aborted by the Master.

IDSEL in

/IDSEL / is used by the Master to perform system configuration. A target will accept a configuration transaction only when /IDSEL is driven high on the 1st cycle. Since every PCI slot has a unique /IDSEL line, every slot can be uniquely accessed even before it's configured[1]. Empty PCI slots can be identified since no /DEVSEL would be driven as a response to a configuration access for that slot.

STOP s/t/s

/STOP is used by the Target to end the current burst read or write transaction. If a Master samples /STOP low during a data word transfer, it must disconnect the bus and end the transaction. The state of /TRDY and /DEVSEL while STOP was driven indicates the termination type: Disconnect with data, Disconnect without data, Target retry , or Target abort.

4.1.5.5 Arbitration

REQ t/s[2]

/REQ is used only by bus Masters. When a Master wants to begin a transaction, it must first acquire the bus by driving /REQ low. All the /REQ lines from all the PCI slots supporting bus masters are connected to a central PCI Bus Arbiter, which will grant the bus only to one PCI Master at a time.

GNT t/s

/GNT is used only by bus Masters. /GNT is driven low by the central PCI Bus Arbiter to the PCI Bus Master when the Master is granted the use the shared PCI bus. /GNT lines are unique to every PCI slot supporting bus Masters. When a Bus Master receives /GNT low, it is allowed to access the PCI bus

---

[1] The PCI standard does **not** define a **standard** way in which a Master can drive a specific **IDSEL** line! On **most** PC based platforms, each slot has its **IDSEL** line connected through a resistor to one of the AD lines, usually beginning with **AD16** for the 1st slot. A Master will select the specific slot configuration space by making sure it drives an address enabling only a single card with **IDSEL**. Driving more than one card with **IDSEL** will cause bus clashes when both cards drives **DEVSEL** , trying to accept the cycle, and may even lead to hardware malfunctions.

[2] Notice that both **REQ** and **GNT** are marked as **t/s**. This is because the signals must be tri stated when the PCI bus is reset.

only after the current cycle taking place is ended. This is indicated by both /FRAME and /IRDY being high.

4.1.5.6 Error Reporting

PERR s/t/s

/PERR is used by a PCI card to report parity errors on the data phase. /PERR is driven by the Master during a data read, and by the Target during a data write. /PERR timing is one cycle after PAR is driven with a valid value. (PAR is driven by the Target during a data read, and by the Master during a data write). Reporting parity error via /PERR is optional, and can be turned off by clearing a control bit in the Target or Master configuration space. Also notice that address phase parity errors should not be reported with /PERR.

SERR o/d

/SERR is used by the Target or the Master to signal a system error condition, such as parity error during a transaction address phase (The 1st cycle, when the address is transmitted). Unlike /PERR, /SERR can be driven by any PCI target at any time since it is open drain. It is pulled high by a resistor located on the motherboard, or on another central resource. /SERR should be used with care, since on most system today it is meant to indicate a fatal system error which might imply a system reset!

4.1.5.7 64 Bit Extension

The following signals exist only on 64 bit PCI cards or slots. The signals are available on a secondary connector, similar to the ISA 16 bit extension of the XT bus. Using 64 bit cards on 32 bit slots (in 32 bit mode of course) is permitted, as well as using 32 bit cards in 64 bit slots.

AD[63:32] t/s

AD[63:32] are used to hold the extra 32 data bits during 64 bit data transfer, and to hold the extra 32 address bits when accessing a resource located in the 64 bit memory space.

C/BE [7:4] t/s

During the address phase of a 64 bit PCI transaction, C/BE [7:4] are unused. During the data phase, the C/BE [7:4] lines are used just like the C/BE [3:0] lines, indicating which byte lanes are valid.

PAR64 t/s

/PAR64 is used in the same was as PAR, but contains the parity only for the AD[63:32] and C/BE [7:4] lines. It obeys the same timing rules as PAR.

REQ64 s/t/s

/REQ64 is used by a 64 bit master to request a 64 bit cycle. It obeys the same timing rules as the /FRAME signal.

ACK64 s/t/s

/ACK64 is used by the target to accept a request for a 64 bit cycle. It obeys the same timing rules as the /DEVSEL signal.

4.1.5.8 Bus Snooping

The bus snooping lines allows a PCI bus master containing a write back cache to maintain the data coherency on systems with multiple bus masters. For example, Bus master “A” contains a modified copy of memory location “X” on his write back cache. In the same time, other bus masters may independently try to modify memory location “Y”, sharing the same cache line as memory location “X”. Since the cache logic writes back to memory whole cache lines, Bus master “A” must write back it's modified copy of the cache line before the other master can write location “Y”, because otherwise any new value for “Y” would be overwritten when Bus master “A” would write back its modified cache line containing “X”.

SDONE in/out[3]

SDONE is used by a bus master containing a write back cache to signal the currently addresses target not to acknowledge the current cycle until the master finishes searching for the specified address in its write back cache. SDONE Will be high during the search, and low on the 1st cycle the search is completed.

SBO in/out

/SBO is driven high during the snoop address search (see above). If the address requested is cached in the local write back cache and is dirty, SBO will go low on the same cycle SDONE goes high, to signal a back-off command to the target, so the modified cache line can be flushed. If /SBO goes high when SDONE goes high, the requested address is not in the master’s cache, and the cycle can continue normally.

[3] **SDONE** and **SBO** are **in** for PCI targets and **out** for PCI masters.

4.1.5.9 JTAG (IEEE 1149.1)

JTAG is an IEEE standard defining a serial bus for system testing. JTAG is used by many chip vendors for:

1. Board testing A PCB connectivity netlist can be verified by bypassing the output and input logic on the chip's I/O pads and drive out test values. By chaining all the chips on a PCB with JTAG, it is possible to test the PCB connectivity after manufacturing or during maintenance by driving values through an I/O pin using the JTAG port of the source chip, and reading those values on another I/O pin using the JTAG port on the target chip.

2. Software debugging of Microprocessors Most new microprocessors supports JTAG for the use of a background debug mode. This mode allows reading and writing internal CPU registers, inserting breakpoints, single stepping the CPU and even get a limited back trace. (For example, some members of the Motorola 683XX family). This gives designers most of the capabilities of an expensive ICE at fraction of the price.

3. In-System-Programming of programmable logic devices

Most of the large pin count FPGAs and CPLDs today use PQFP packages which are soldered directly to the board during manufacturing. When using multiple chips it is very expensive to program these chips on a programmer. It is also impossible to reprogram the devices once they are soldered. Instead, CPLDs and FPGA chips are soldered to the board while they are still blank, and are chained together on the board by a JTAG chain. A single JTAG connector on the board can then be used to program all the chips without using an expensive and unreliable PQFP adapter for off-board chip programming. Unfortunately, the PCI committee never defied a standard way to access the JTAG port on the PCI motherboard, nor did it define the JTAG bus topology. Because of this, only a few PCI designs supported the JTAG pins on the PCI connector. Some motherboard manufacturers has even went as far as violating the PCI spec and using the JTAG lines for an entirely different purpose such as Video sideband signals!

TDI

TDI is a Test Input pin, and it is used to shift data and instructions into the JTAG port.

TDO

TDO is used to shift data out from the JTAG port.

TCK

TCK is the JTAG clock. It is used to control the data shifting in and out of the JTAG port.

TMS

TMS is used to select the JTAG mode.

TRST

TRST is used to reset the JTAG port.

4.1.6 The PCI Bus Commands

As we have seen in section 4.1.5.3, the PCI Bus has defined 12 different commands and reserved 4 commands. The following paragraph contains a very short description of all the PCI Bus commands. We will take a closer look at the commands later, in section 4.1.9.

Memory Read

A Memory Read command is used by a PCI bus master to read one or more memory locations from a PCI bus target. It is recommended that Masters use this command when reading less than one cache line.

Memory Read Line

This command is the same as the Memory Read command, with the exception that the bus master is intending to read at least a cache line. This is a hint to the Target, which may use this hint to prefetch more data in advance, or it may choose to ignore it and treat this command in the same way as the Memory Read command. It is recommended that Masters use this command when reading between 1 to 2 cache lines.

Memory Read Multiple

This command is the same as the Memory Read command, with the exception that the bus master is guaranteed to read one or more cache lines. This is a hint to the target, which may use this hint to prefetch more data in advance, or it may choose to ignore it and treat this command in the same way as the Memory Read command. It is recommended that Masters use this command when reading 2 or more cache lines.

Memory Write

A Memory Write command is used by a PCI Bus master to write one or more memory locations to a PCI target.

Memory Write and Invalidate

This command is the same as the Memory Write command, with the exception that the Master is guaranteed to write one or more whole cache lines. This command is used to disable the snooping mechanism, because writing an entire cache line means that any dirty data located in any cache is now invalid and shouldn't be written back. When using this command, the master must never write incomplete cache lines.

I/O Read

An I/O Read command is used by a PCI bus master to read the one or more I/O locations from a PCI target.

I/O Write

An I/O Write command is used by a PCI bus master to write one or more I/O locations to a PCI target.

Configuration Read

A Configuration Read command is used by a PCI bus master to read one or more Configuration registers from a PCI target.

Configuration Write

A Configuration Write command is used by a PCI bus master to write one or more Configuration registers to a PCI target.

Interrupt Acknowledge

This command is used by X86 platforms to pass the interrupt acknowledge cycles needed when a CPU communicates with a 8259 Programmable Interrupt Controller. The X86 family chips has only one INTR line, and the PCI is using the Interrupt Acknowledge cycle to notify the CPU which IRQ line has been activated. In most PCI based PC motherboards, the PCI chips are NOT on the PCI Bus, and all the Interrupt Acknowledge cycles appear on the CPU local bus and not on the PCI Bus.

Special Cycle

The Special Cycle command is used to send an event to all the PCI Bus targets. The event is identified by an event code, which is allocated by the PCI SIG. Predefined events includes the X86 shutdown and halt events. Unlike other cycles, Special cycles are not acknowledged by the PCI bus targets, i.e. DEVSEL is not asserted for this cycle.

Dual Address Cycle

A Dual Address Cycle is used to access a 64 bit address on 32 bit PCI slots. A Dual Address Cycle command is sent together with the upper 32 address bits on AD[31:0] followed by the required command (Memory or I/O R/W).

4.1.7 The PCI Address structure

The PCI bus recognizes 3 types of address spaces: Memory, I/O and Configuration. In the following paragraph we will discuss these addresses spaces and describe the way an address is constructed for the different PCI commands.

4.1.7.1 Memory address space

The Memory address space is the main address space used by PCI cards. A PCI memory address is 32 bit wide, or optionally, 64 bit wide. 64 bit addresses may be generated either by a 64 bit wide PCI bus master and 64 bit wide PCI target (plugged on a 64 bit wide PCI backplane!), but can also be generated in a 32 bit PCI slot when the Dual Address Cycle command is used. The PCI SIG recommends all the resources occupied by a device to be mapped into the PCI memory space.

During memory read/write commands, the address is interpreted according to the following rules:

AD[31:2] Longword address in the 4GB Memory address space. Since only longword addresses are allowed, target address are always aligned on a longword boundary, i.e. it is impossible to have one target on address 1000h, and a different target on address 1001h.

AD[1:0] Address increment mode, which defines how the next address in a burst sequence is calculated.

The address increment modes are indicated the following table 4.3:

AD[1:0]	Mode	Description
00	Linear increment mode	The next word's address in the burst will be the next sequential address in memory after the current address.
01	Reserved	
10	Cacheline wrap mode	Same as Linear increment mode, but when the cacheline boundary is crossed, the next address wraps around to the beginning of the cacheline. When

		the burst length reaches the cacheline size, the next address is incremented by the cache line size into the same cacheline position in the next cacheline.
11	Reserved	

Table 4.3 PCI addressing mode for memory read/write commands

4.1.7.2 I/O address space

The PCI I/O address space is 32 bit wide, but most system does not support more than 16 bits of I/O addresses. The PCI I/O address space is used mostly for backward compatibility with legacy hardware on X86 based systems, and is not recommended for new designs. On most systems, I/O access is slower than memory access, and I/O space is severely limited in the PC architecture.

Note: On PCI implementations where no I/O instruction exists, I/O space cycles may be generated by a special mechanism, such as mapping part of the I/O space into a predefined memory area.

During I/O read/write commands, AD[31:0] contains the byte address in the 4GB I/O space. It is possible to have I/O targets on a byte boundary. This is really important when more than one peripheral may share addresses on a word boundary. Since data is still transferred by longwords, a target must check the individual byte enables and disconnect when a Master tries to access bytes outside the Target's address space.

For Example:

Master drives an I/O write transaction with AD[31:0] = 379h. If the next data word is written with C/BE [3:0] = 0000b, it means bytes 378h to 37bh are meant to be written. Since 378h is below the requested Target I/O address, it must disconnect without accepting the data!.

This can be summarized in the next table:

AD[1:0]	C/BE [3]	C/BE [2]	C/BE [1]	C/BE [0]
00b	X	X	X	X
01b	X	X	X	1
10b	X	X	1	1
11b	X	1	1	1

Table 4.4 Legal CBE [3:0] and AD[1:0] combinations for I/O read/write commands

4.1.7.3 Configuration address space

The configuration address space is a very small memory area, 256 bytes long, that is unique for every PCI bus/device/function triplet in the system. The PCI standard defines a standard header that uses the first 64 bytes of the header, and keeps the remaining 192 bytes user defined. Configuration space is accessed when a configuration command is sent to a card while its IDSEL line is active. Unlike other bussed PCI signals, IDSEL is uniquely driven into each card (a star topology), so when a card is selected by an active IDSEL signal, it is the only active card in that cycle.

During configuration read/write commands, the address field can be interpreted in one of two ways.

Configuration Type 0 is issued from a master which is on the same bus as the target. Configuration Type 1 is issued by a PCI Master trying to access configuration space on a PCI device hidden behind one or more PCI to PCI bridges. Type 1 requests are intercepted by bridge devices, which may turn it into a Type 0 request on the secondary bus, or forward it if the secondary bus is not the final destination bus. **Configuration Type 0:**

AD[31:11] These lines are ignored. As we mentioned above, AD[31:16] are used in some designs to generate IDSEL.

AD[10:8] Function Number. A single PCI card may incorporate multiple functions, each with it's own configuration space. Since each function has a separate configuration space, it can be configured by a separate driver. This is very important when multiple legacy cards are integrated on a new card, and it is desired to keep the old drivers (for software compatibility).

AD[7:2] Configuration register number. This specifies one of 64 Long words. Only the first 16 registers are defined by the PCI standard. The rest are user defined.

AD[1:0] Always 00b.

Configuration Type 1:

AD[31:11] Reserved.

AD[23: 16] Bus Number.

AD[15:11] Device Number.

AD[10:8] Function Number.

AD[7:2] Configuration register number.

AD[1:0] Always 01b.

4.1.7.4 The PCI configuration header

The PCI configuration contains information about the PCI card in a particular slot. Some of these registers are read only, and some are read/write.

31 16	15 0	
Device ID	Vendor ID	00h
Status	Command	04h
Class Code	Revision ID	08h
BIST / Header / Type Latency Timer	Cache Line Size	0Ch
Base Address Register 0		10h
Base Address Register 1		14h
Base Address Register 2		18h
Base Address Register 3		1Ch
Base Address Register 4		20h
Base Address Register 5		24h
CardBus CIS Pointer		28h
Subsystem ID	Subsystem Vendor ID	2Ch
Expansion ROM Base Address		30h
Reserved		34h
Reserved		38h
Max_Lat / Min_Gnt / Interrupt Pin	Interrupt Line	3ch

Table 4.5 PCI configuration header 0

Here is a short list of some of the fields in this header:

Vendor ID

This read-only field contains a unique Vendor ID identifying the designer/manufacturer of this PCI chip. PCI vendor ID codes are assigned by the PCI SIG (Special Interest Group) for SIG members.

Device ID

This read-only field contains a unique Device ID for this PCI device. This field is allocated by the specific vendor designing/manufacturing the device. An

operating system could use the Device ID and Vendor ID fields to select a specific device driver for this chip.

Command

This read/write field contains a 16 bit command register. Some of the bits in this register are reserved, and some are optional. Bits which are not implemented, must be read-only, and return 0 on read. Here is a short explanation of all the different bits in this register in table 4.6.

Bit	Location Description
0	I/O space enable. Writing 1 enables the device response to I/O space access. Defaults to 0 after RST . If device has no I/O, this bit must be hardwired to 0.
1	Memory space enable. Writing 1 enables the device response to Memory space access. Defaults to 0 after RST . If the device is not memory mapped, this bit must be hardwired to 0.
2	Master Enable. Writing 1 enables bus master behavior. Defaults to 0 after RST . If this is a target only device, this bit must be hardwired to 0.
3	Special Cycle Enable. Writing 1 enables response to special cycles. Defaults to 0 after RST . If special cycles are not used, this bit must be hardwired to 0.
4	MWI Enable. Writing 1 allows a bus master to use the Memory Write and Invalidate command. Writing 0 forces the use of Memory Write command. Defaults to 0 after RST . If the master does not support MWI, this bit must be hardwired to 0.
5	VGA Palette snooping enable. When 1, device must snoop VGA palette access (i.e. snoop data on bus, but do not respond). When 0, treat like normal addresses. If device is not VGA compatible, this bit must be hardwired to 0.
6	Parity error enable. When 0, Ignore parity errors. When 1, do normal parity error action. If parity checking not supported, this bit must be hardwired to 0. Devices that do not check for parity errors must still generate correct parity.
7	Address/Data stepping enable. If device does address/data stepping,

	this bit must be hardwires to 1. If device does not do address/data stepping, this bit must be hardwires to 0. If device can optionally do address/data stepping, this bit must be read/write, and default to 1 after RST ..
8	SERR Enable. When 1, SERR generation is enabled. When 0, no SERR is generated. Defaults to 0 after RST . Can't be hardwired to 0 if SERR not supported.
9	Fast back to back Enable. When 1, master is allowed to generate fast back to back cycles to different targets. When 0, master is allowed to generate fast back to back cycles only to the same targets. This bit is initialized by the BIOS to 1 if all targets are fast back-to-back capable. Defaults to 0 after RST .
15:10	Reserved. Must return 0 on read.

Table 4.6 PCI Command configuration register

Status

This read-only field contains a status register. Some of the bits in this register are reserved, and some are optional. Bits which are not implemented, must return 0 on read. Here is a short explanation of all the different bits in this register in table 4.7.

Bit	Location Description
4:0	Reserved. Must return 0 on read.
5	66MHz Capable. Return 1 if device is 66MHz Capable, 0 if only 33MHz supported.
6	UDF Supported. Return 1 if adapter requires UDF configuration file. Return 0 for normal devices.
7	Fast Back to Back Capable. Return 1 if target is able to accept fast back to back transactions when the transactions are not to the same target.
8	Data parity Error Detected. Implemented by PCI masters only. True if parity error detected, PERR set, parity error response bit is set.
10:9	DEVSEL Timing. 00 for Fast, 01 for Medium, 10 for Slow, 11 is reserved. These bits must represent the slowest timing for all commands except for configuration read and write.

11	Signaled Target Abort. Set by a target when it terminated a cycle with a target abort.
12	Received Target Abort. Set by a master when it detected a cycle ending with a target abort. All masters must implement this bit.
13	Received Master Abort. Set by a master when it detected a cycle ending with a master abort. All masters must implement this bit.
14	Signaled System Error. Set if device asserted SERR . Must be implemented only by device capable of generating SERR .
15	Detected parity Error. Set if parity error detected. Must be set on error even if parity error handling (bit 6) is disabled.

Table 4.7 PCI Status configuration register

Revision ID

This read-only field contains a unique Revision ID for this PCI device. This field is used to identify different versions of the same chip. For example, if different revisions of the chip contained different bugs, a device driver software could use the Revision ID field to select different workarounds to bypass different chip bugs.

Class Code

This read-only field contains a field describing the device type. Some of the device types identify specific types of devices which can then be programmed by a generic device driver (for example, IDE interface, VGA card, PCI to PCI bridge). Some other codes describe a device type, but still require a specific device driver. (A SCSI controller, for example). For a list of class codes please see Appendix .

Cache Line Size

This read/write field is filled by the BIOS or the operating system by the CPU cache line size. This information is used by master and target devices if they support burst transfers in cacheline wrap mode.

Latency Timer

This read/write field is used to initialize the PCI master latency timer. The latency timer is initialized each time the PCI master device is granted the bus, and begins a countdown when its GNT line is deasserted. When the latency timer expires, the master must terminate its burst. This field can be read-only

for masters that does not support burst longer than two words, but must be less than 16. Actually, not all the bits of this register must be implemented. One or more bits (beginning with the least significant bit) can be made to return 0 instead, reducing the timer's granularity.

Header Type

This read-only field contains a unique header type code in bits 6:0. Currently defined values are 0 for the standard PCI header, 1 for PCI to PCI bridge header, and 2 for PCI to CardBus bridge header. Header type codes of 3 and above are reserved. The header type controls the layout of the configuration addresses in the range 10h to 3Fh.

Bit 7 of the header type field specifies whether this device is a multi function device. A value of 1 denotes a multi function device.

BIST

This read/write field controls Built-In Self Test for devices supporting this feature. If not used, this field must contain 0.

Bit	Name	Name Description
7	BIST Capable	Return 1 if device supports BIST, 0 otherwise.
6	Start BIST	Write 1 to start BIST. Device will reset bit back to 0 when BIST complete.
5:4	Reserved	Must return 0
3:0	Completion	Code Return 0 when completes with no errors. Non-zero values are device specific errors.

Table 4.8 PCI BIST configuration register

Base Address Register 0 *through* Base Address Register 5

These read/write fields are initialized by the BIOS or operating system by writing the desired base address where the PCI device is to be located in the PCI memory map. A PCI device may use one or more base address registers.

CardBus CIS Pointer

This read-only field is used by devices which are both PCI and CardBus compatible. It is used to point to the Card Information Structure for the CardBus card. Subsystem Vendor ID, Subsystem ID These read-only fields are used to identify a specific product using a generic PCI chip. This field can be used by device drivers to take advantage of specific features in some cards using generic PCI chips. Subsystem Vendor ID values are assigned by

the PCI SIG. Subsystem ID values are unique for every Subsystem Vendor ID. This field is optional in PCI 2.1, but required according to Microsoft's PC97 and PCI 2.2 (to be released soon). It must contains 0 if not used. When implemented by a PCI chip, there is usually a way to set this field externally, most of the time through an external memory chip connected to the PCI device. These way PCI board vendors can set this without making their own PCI device.

Expansion ROM base Address

This is a special type of Base Address Register pointing to the card's expansion ROM. Expansion ROMs are optional, and may contain additional information about the card, boot code (in more than one executable format, as well as the architecture independent OpenBoot format).

Interrupt Line

This read-write field is initialized by the BIOS or Operating system, and will contain a system specific value identifying which hardware interrupt has been assigned to this card. This value is not used by the card itself, but rather by the driver and operating system which reads this field to determine which interrupt vector has been assigned for the device. This field is not used by devices not using interrupts.

Interrupt Pin

This read-only field defines which interrupt pin is used by this device. The values contained this field are 1 to 4, corresponding to INTA to INTD . Devices not using interrupts should put 0 in this field.

Min_Gnt, Max_Lat

These read-only fields are used by the BIOS to calculate the desired value for the latency timer of this device. The Min Gnt field measures, in 250us units, the minimum burst length this device requires. The Max_Lat field describes, in 250us units, how often the device requires access to the PCI Bus.

4.1.8 Basic PCI Cycles

4.1.8.1 PCI Memory or I/O Read Cycle

The following timing diagram demonstrates a typical PCI memory or I/O burst read transaction.

1. At Clock 0 the Master begins the transaction by driving FRAME low, the requested address on AD[31:0], and the requested command on C/BE

[3:0] (I/O Read, Memory Read, Memory Read Line, or Memory Read Multiple).

2. At Clock 1 the Target responds by asserting DEVSEL low. Since clock 1 is used for bus turnaround on read transaction (Master stops driving AD[31:0], Target starts driving AD[31:0]), actual data transfer must begin only on cycle 2, so TRDY and IRDY must be high, since no data is available yet.

3. At clock 2 both IRDY and TRDY are low, so the 1st data word is read (D0).

4. At clock 3 the Master is not ready to accept the new word, since IRDY is high. The Target is ready to transfer the next word since TRDY is low. Since not both are low, no data is transferred.

5. At clock 4 the master is able to receive data again, since IRDY is low, and the 2nd data word is transferred. Since FRAME was high during this last cycle, both the Master and the Target end the cycle.

6. At clock 5 the Target tri-states AD[31:0] (turnaround), and drives TRDY high. The Master also drives IRDY high. This is done since IRDY and TRDY are sustained-tri-state lines, and must be driven high for one cycle prior to tri-stating them. This is done in order to charge the line, which will cause it to stay at a high level even after it is tri stated.

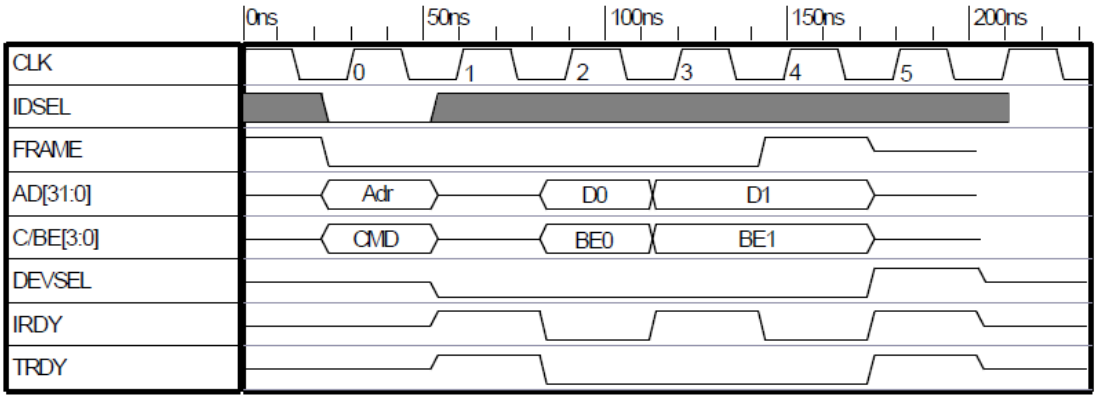


Figure 4.5 *PCI Memory and I/O read cycle*

4.1.8.2 PCI Memory or I/O Write Cycle

The following timing diagram demonstrates a typical PCI memory or I/O burst read transaction.

1. At Clock 0 the Master begins the transaction by driving FRAME low, the requested address on AD[31:0], and the requested command on C/BE [3:0] (I/O Write, Memory Write, or Memory Write Line).
2. At Clock 1 the Target responds by asserting DEVSEL low. Since this is a write transaction, the master keeps driving AD[31:0] (this time with data), and no bus turnaround cycle is needed. IRDY is driven low to indicate data is ready for write. Since the Target is driving TRDY high, it is not ready to accept the data. C/BE [3:0] are loaded with the appropriate byte enables.
3. At clock 2 both IRDY and TRDY are low, so the 1st data word is written (D0).
4. At clock 3 the Master is not ready to send a new word (as indicated by the invalid content of AD[31:0]), so IRDY is driven high. The Target is ready to accept the next word since TRDY is low, but since IRDY is high, no data transfer takes place.
5. At clock 4 the master is able to write data again, since IRDY is low, and data is transferred. Since FRAME was high during this last cycle, both the Master and the Target end the cycle.
6. At clock 5 the Target drives **TRDY** and **DEVSEL** high. The Master also drives **IRDY** high. It may even start a new transaction right now (this is called back to back transfers, when there are no turnaround cycles between multiple transactions. Back-to-back transfers can only happen after a write transaction, when no turnaround cycles are needed).

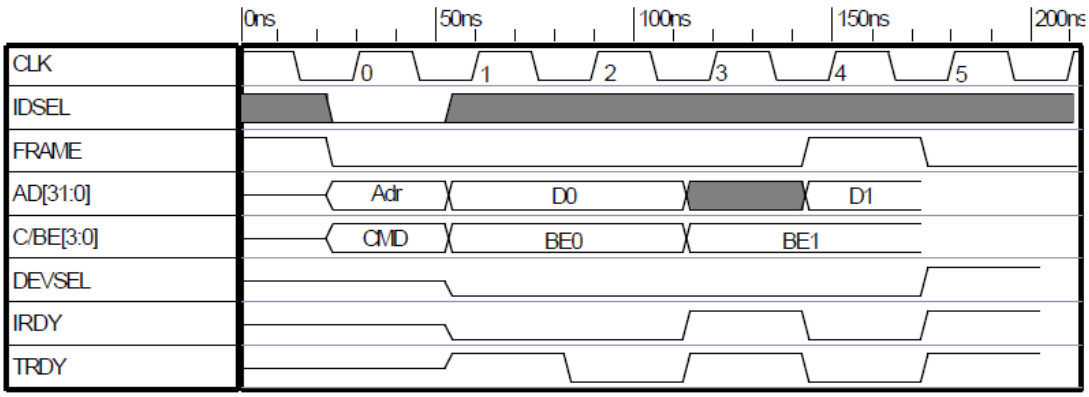


Figure 4.6 PCI Memory and I/O write cycle

4.1.8.3 PCI Configuration Read Cycle

The following timing diagram demonstrates a typical PCI configuration type 0 read transaction. Since no address increment is defined for configuration cycles, burst behavior is undefined for configuration space, and therefore not used. Usually, the target will disconnect after the 1st word. Configuration type 1 cycles do not use IDSEL, and are identical to ordinary read cycles, but do not support bursts, like type 0 configuration cycles.

1. At Clock 0 the Master begins the transaction by driving FRAME low, IDSEL high, the requested address on AD[10:0], and the requested command on C/BE [3:0] (Configuration Read).

2. At Clock 1 the Target responds by asserting DEVSEL low. Since clock 1 is used for bus turnaround on read transaction (Master stops driving AD[31:0], Target starts driving AD[31:0]), actual data transfer must begin only on cycle 2, so TRDY and IRDY must be high, since no data is available yet.

3. At clock 2 both IRDY and TRDY are low, so the 1st data word is read (D0). Since FRAME was high during this last cycle, both the Master and the Target end the cycle.

4. At clock 3 the Target tri-states AD[31:0] (turnaround), and drives TRDY high. The Master also drives IRDY high. This is done since IRDY and TRDY are sustained-tri-state lines, and must be driven high for one cycle prior to tri-stating them. This is done in order to charge the line, which will hold it at a high level even after it is tri stated.

5. At clock 4, all the sustained tri state lines driven high before (TRDY, IRDY ,DEVSEL , FRAME) are now tri-stated.

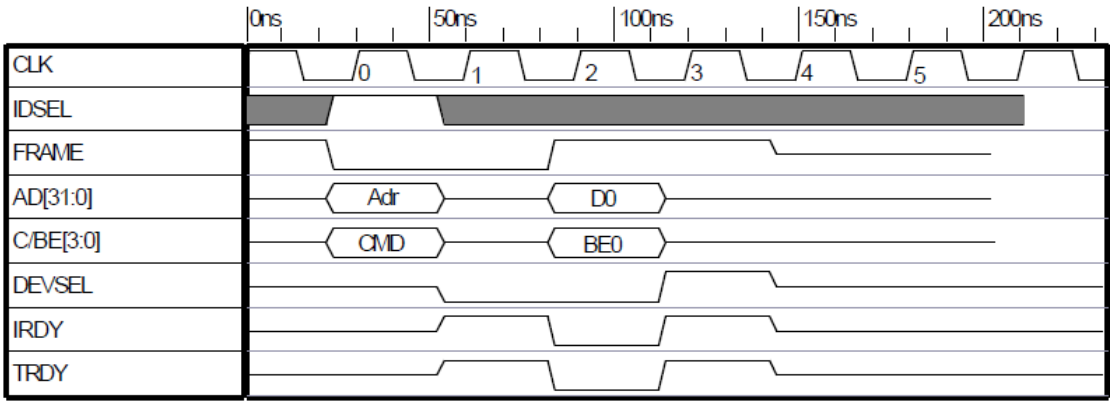


Figure 4.7 PCI Configuration Read cycle

4.1.8.4 PCI Configuration Write Cycle

The following timing diagram demonstrates a typical PCI configuration write transaction. Since no address increment is defined for configuration cycles, burst behavior is undefined for configuration space, and therefore not used. Usually, the target will disconnect after the 1st word.

Configuration type 1 cycles do not use IDSEL, and are identical to ordinary write cycles, but do not support bursts, like type 0 configuration cycles.

1. At Clock 0 the Master begins the transaction by driving FRAME low, IDSEL high, the requested address on AD[10:0], and the requested command on C/BE [3:0] (Configuration Write).

2. At Clock 1 the Target responds by asserting DEVSEL low. IRDY is driven low because the master is ready, but TRDY is high, because the target is not ready yet. FRAME is driven high since this is the last word for this burst.

3. At clock 2 both IRDY and TRDY are low, so the 1st data word is written (D0). Since FRAME was high during this last cycle, both the Master and the Target end the cycle.

4. At clock 3 the Target tri-states AD[31:0] (turnaround), and drives TRDY high. The Master also drives IRDY high. This is done since IRDY and TRDY are sustained tri-state lines, and must be driven high for one cycle prior to tri-stating them. This is done in order to charge the line, which will hold it at a high level even after it is tri stated.

5. At clock 4, all the sustained tri state lines driven high before (TRDY, IRDY, DEVSEL, FRAME) are now tri-stated.

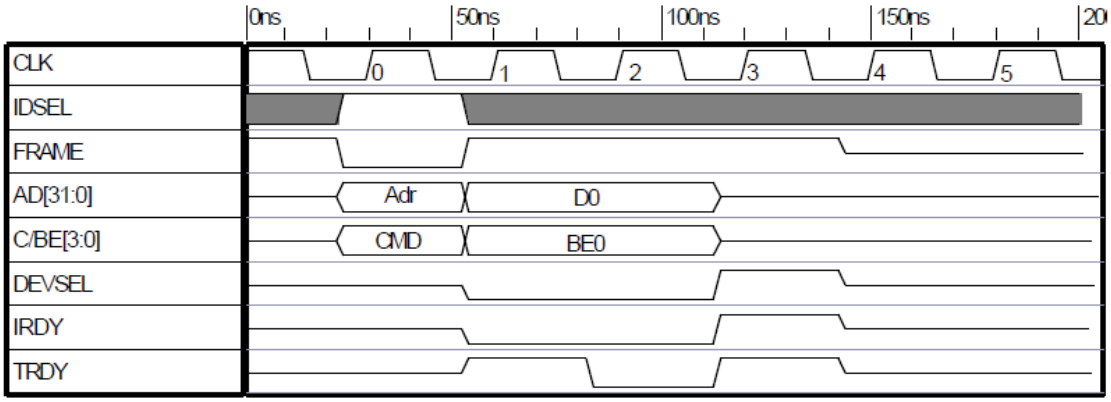


Figure 4.8 PCI Configuration Write cycle

4.1.9 Abnormal cycle termination

The sample cycles we have shown so far were always terminated by the master by driving FRAME high during the last data word transfer. In this section, we will show how PCI cycles can end in different ways. All the figures in this section apply equally to all data transactions types. i.e. memory, I/O or configuration, as well as for read or write transactions. The data bus, on the other hand was drawn from the perspective of a read transaction (a new data word is valid after TRDY is asserted, not when IRDY is asserted), but all the information should equally apply to write transaction as well.

4.1.9.1 Target termination (Disconnect with data)

When a target can no longer sustain a burst, it can assert STOP during the same cycle it asserts TRDY. It must keep STOP asserted until FRAME is deasserted. TRDY is deasserted after IRDY is asserted, to prevent transferring another word. FRAME will be deasserted in the same cycle IRDY is asserted. Target termination is used for a graceful transaction termination without any errors. The master may (or may not) continue the same burst by starting a new transaction at the subsequent address after the last one which was transferred.

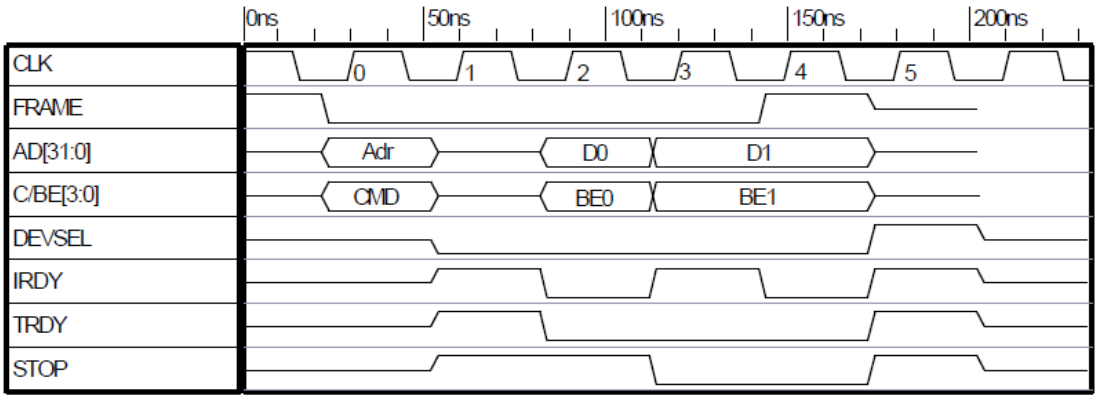


Figure 4.9 Target Disconnect with data

4.1.9.2 Disconnect without data

When a target cannot supply the next word in a burst, it can assert STOP and deasserts TRDY. It keeps STOP asserted until FRAME is deasserted. TRDY is kept deasserted until the end of the cycle, and can be tri-stated when FRAME is high. If FRAME was low, IRDY is must be asserted at the same cycle FRAME is deasserted.

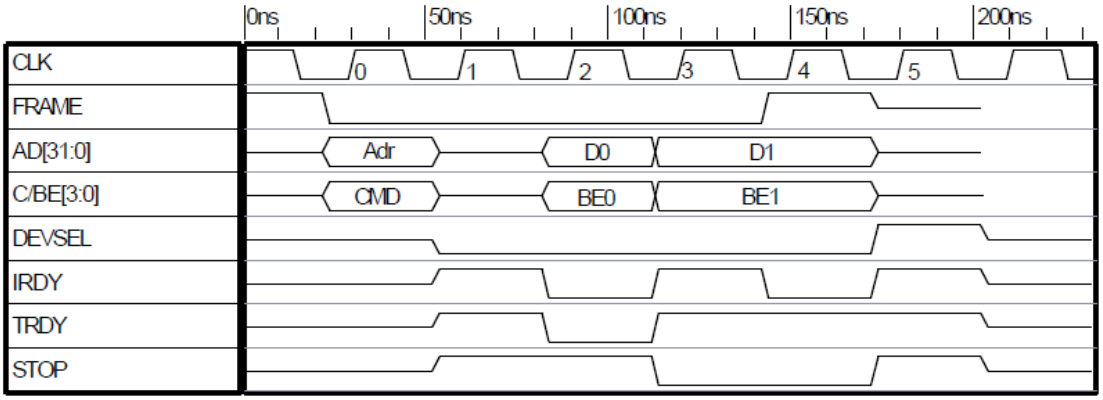


Figure 4.10 Target disconnect without data

4.1.9.3 Target Retry (Disconnect without data)

Target retry is a special case of Disconnect without data occurring at the first word. Target retry is used to signal the master that the target is not ready yet, but will be ready later. The master must retry the same transaction over and over until it succeeds. The master also must release the bus (by deasserting its REQ) for at least two clock cycles between retry attempts to let other bus masters share the bus.

4.1.9.4 Target abort

When a serious error condition has occurred, the target can assert STOP and deasserts DEVSEL and TRDY. It keeps STOP asserted until FRAME is deasserted. TRDY and DEVSEL are kept deasserted until the end of the cycle, and can be tri-stated when /FRAME is high. If /FRAME was low, /IRDY is must be asserted at the same cycle /FRAME is deasserted. Target abort is used to stop a transaction only when the transaction has failed and will never succeed (for example, trying to write a read only memory). The master will not retry the transaction again.

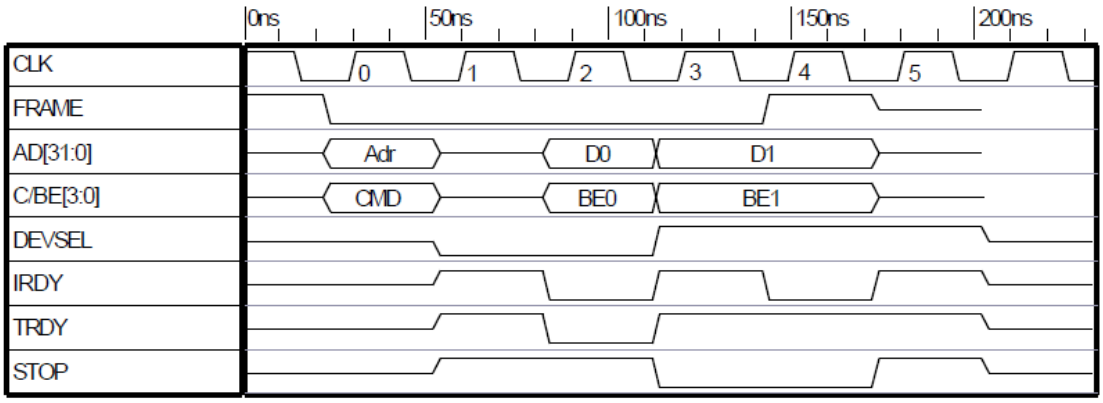


Figure 4.11 target abort

4.1.9.5 Master abort

Another serious error condition occurs when the master begins a cycle, but none of the targets acknowledges it within 4 cycles. When this happens, the master will abort the cycle by deasserting /FRAME and ending the transaction.

4.1.10 The PCI to PCI Bridge

As we already noted, the PCI bus standard is limited to a small number of PCI slots. The main PCI mechanism to expand a PCI system beyond the limited number of slots is the PCI to PCI Bridge. A PCI bridge has two busses, a primary bus and a secondary bus. The two busses can be completely independent, as much as running on two different clock rates. The PCI to PCI bridge appears as both a master and a target to both busses. When accessed as a target from the primary bus, any access falling within the bridge address space would be regenerated on the secondary bus. When accessed as a target on the secondary bus, any address falling outside the secondary bus

address range would be regenerated on the primary bus. Notice that the secondary bus will not respond to configuration type 0 commands.

Very large PCI systems may have up to 255 busses arranged this way. The PCI to PCI Bridge has a slightly different configuration header, called header type 1. The header structure, as well as the register description is listed below. PCI to PCI Bridge are complex devices and we cannot cover all their functions. For a complete specifications of the PCI to PCI Bridge devices, please refer to [3].

4.2 AVR 8 – bit Microcontroller family.

The AVR enhanced RISC microcontrollers [1] are based on a new RISC architecture that has been developed to take advantage of semiconductor integration and software capabilities of the 1990's. The AVR is a family of RISC microcontrollers from Atmel. The AVR architecture, first conceived by two electronic engineering students at the Norwegian Institute of Technology (NTH), has been refined and developed by Atmel Norway, in a division founded by the chip's architects. A block diagram of the AVR architecture is given in figure 4.14. The memory sizes and peripherals indicated in the figure 4.12 are for the ATmega32 microcontroller.

The key benefits of AVR family of microcontrollers are:

- High performance.
- Low power consumption.
- High code density.
- Outstanding memory technology.
- High integration.

The main purpose to use AVR is the scalability of the family. The devices ranges from 1to 256KB, pin count ranges from 8 to 100, full code compatibility, pin/feature compatible family and single set of development tools. Any member is either a subset or superset of other family members of the AVR microcontroller family.

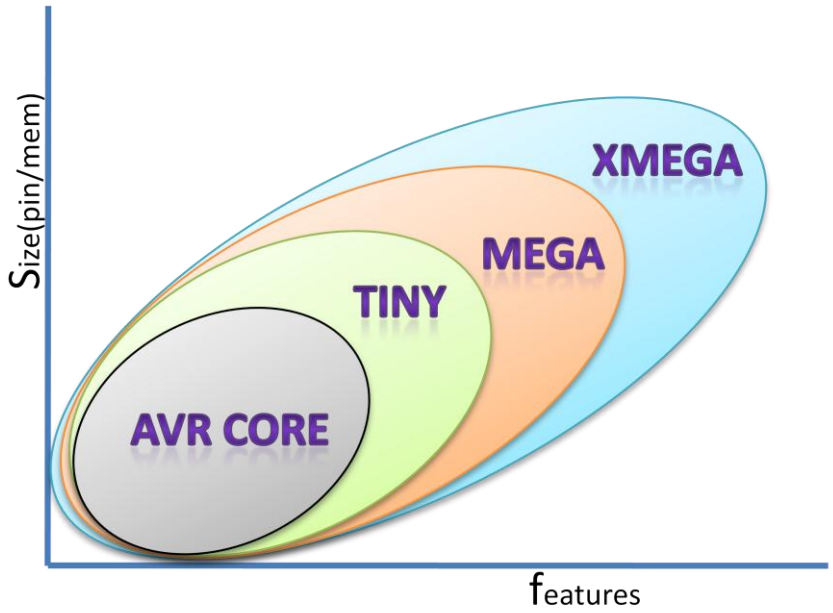


Figure 4.12 AVR Scalability

Central in the AVR architecture is the fast-access RISC register file, which consists of

32 x 8-bit general purpose working registers. Within one single clock cycle, AVR can feed two arbitrary registers from the register file to the ALU, do a requested operation, and write back the result to an arbitrary register. The ALU supports arithmetic and logic functions between registers or between a register and a constant. Single register operations are also executed in the ALU.

As can be seen from the figure, AVR uses a Harvard architecture, where the program memory space is separated from the data memory space. Program memory is accessed with a single level pipelining. While one instruction is being executed, the next instruction is being pre-fetched from the program memory. Due to the true single cycle execution of arithmetic and logic operations, the AVR microcontrollers achieve performance approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

High Level Languages (HLLs) are rapidly becoming the standard methodology for embedded microcontrollers due to improved time-to-market and simplified maintenance support. In order to ensure that the new ATMEL

AVR family of microcontrollers was well suited as a target for C compiler, the external C compiler development was started before the AVR architecture and instruction set were completed. During the initial development of the C compiler, several potential improvements in the AVR were identified and implemented. The result of this cooperation between the compiler developer and the AVR development team is a microcontroller for which highly efficient, high performance code is generated.

About ATmega32

The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega32 provides the following features: 32K bytes of In-System Programmable Flash Program memory with Read-While-Write capabilities, 1024 bytes EEPROM, 2K byte SRAM, 32 general purpose I/O lines, 32 general purpose working registers, a JTAG interface for Boundary-scan, On-chip Debugging support and programming, three flexible Timer/Counters with compare modes, Internal and External Interrupts, a serial programmable USART, a byte oriented Two-wire Serial Interface, an 8-channel, 10-bit ADC with optional differential input stage with programmable gain (TQFP package only), a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and six software selectable power saving modes. The Idle mode stops the CPU while allowing the USART, Two-wire interface, A/D Converter, SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next External Interrupt or Hardware Reset. In Power-save mode, the Asynchronous Timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping.

The ADC Noise Reduction mode stops the CPU and all I/O modules except Asynchronous Timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running

while the rest of the device is sleeping. This allows very fast start-up combined with low-power consumption. In Extended Standby mode, both the main Oscillator and the Asynchronous Timer continue to run. The device is manufactured using Atmel's high density nonvolatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed in-system through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega32 is a powerful microcontroller that provides a highly-flexible and cost-effective solution to many embedded control applications.

The ATmega32 AVR is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, in-circuit emulators, and evaluation kits.

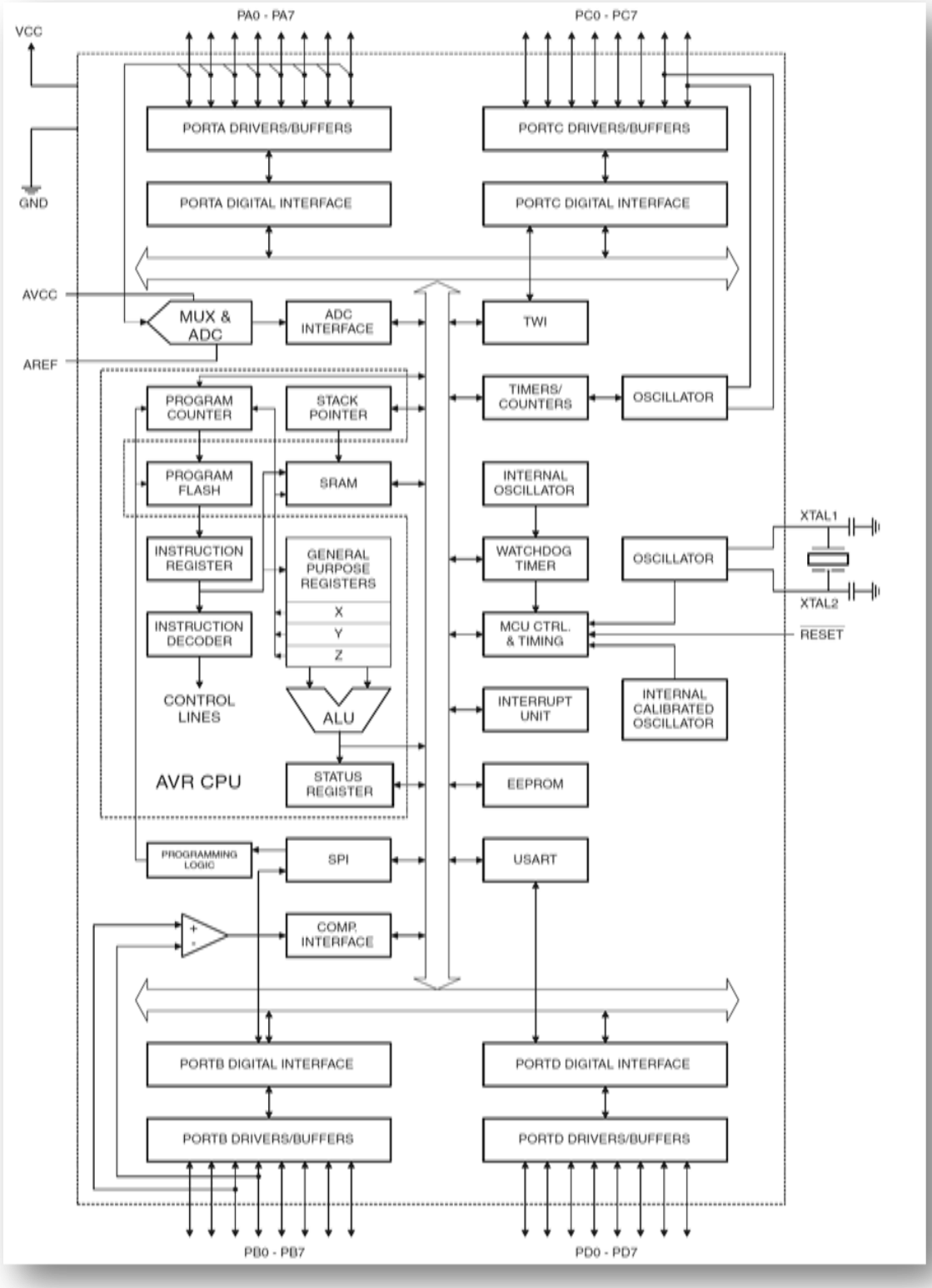


Figure 4.13 Block Diagram of ATmega32

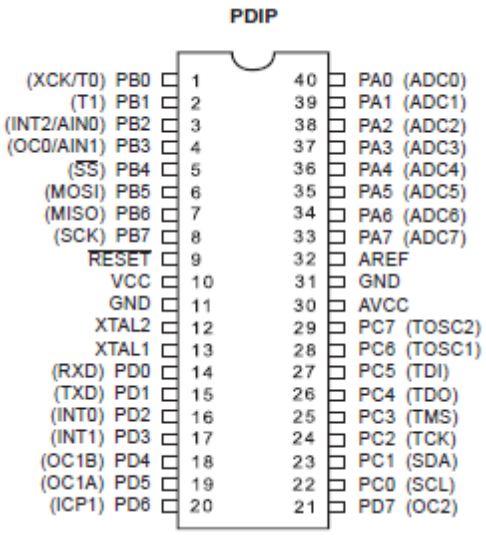


Figure 4.14 Pinout of ATmega32

AVR CPU Core

Introduction This section discusses the AVR core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupt

Architectural Overview

In order to maximize performance and parallelism, the AVR uses a Harvard architecture with separate memories and buses for program and data. Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is In-System Reprogrammable Flash memory.

The fast-access Register File contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two operands are output from the Register File, the operation is executed, and the result is stored back in the Register File – in one clock cycle.

Six of the 32 registers can be used as three 16-bit indirect address register pointers for

Data Space addressing – enabling efficient address calculations. One of these address pointers can also be used as an address pointer for look up

tables in Flash Program memory. These added function registers are the 16-bit X-, Y-, and Z-register, described later in this section.

The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the Status Register is updated to reflect information about the result of the operation.

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction.

Program Flash memory space is divided in two sections, the Boot program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM instruction that writes into the Application Flash memory section must reside in the Boot Program section.

During interrupts and subroutine calls, the return address Program Counter (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM, and consequently the Stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the reset routine (before subroutines or interrupts are executed). The Stack Pointer SP is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture.

The memory spaces in the AVR architecture are all linear and regular memory maps. A flexible interrupt module has its control registers in the I/O space with an additional global interrupt enable bit in the Status Register. All interrupts have a separate interrupt vector in the interrupt vector table. The interrupts have priority in accordance with their interrupt vector position. The lower the interrupt vector address, the higher the priority.

The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers, SPI, and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the Register File, $20 - $5F.

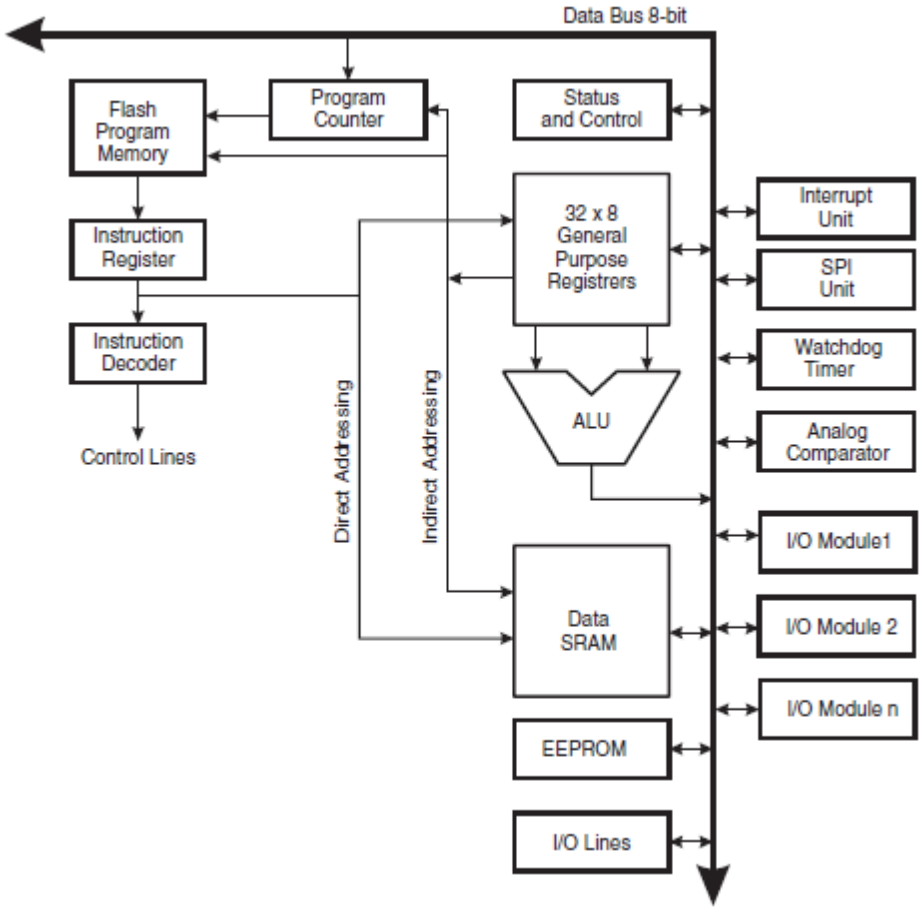


Figure 4.15 Block Diagram of the AVR MCU Architecture

JTAG Interface and On-chip Debug System

Features

• JTAG (IEEE std. 1149.1 Compliant) Interface

• Boundary-scan Capabilities According to the IEEE std. 1149.1 (JTAG) Standard

• Debugger Access to:

– All Internal Peripheral Units

– Internal and External RAM

– The Internal Register File

– Program Counter

– EEPROM and Flash Memories

– Extensive On-chip Debug Support for Break Conditions, Including

– AVR *Break* Instruction

– Break on Change of Program Memory Flow

– Single Step Break

– Program Memory Breakpoints on Single Address or Address Range

– Data Memory Breakpoints on Single Address or Address Range

• Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface

• On-chip Debugging Supported by AVR Studio®

Overview The AVR IEEE std. 1149.1 compliant JTAG interface can be used for

• Testing PCBs by using the JTAG Boundary-scan capability

• Programming the non-volatile memories Fuses and Lock bits

• On-chip Debugging

Boot Loader Support – Read-While-Write Self-Programming

The Boot Loader Support provides a real Read-While-Write Self-Programming mechanism for downloading and uploading program code by the MCU itself. This feature allows flexible application software updates controlled by the MCU using a Flash-resident Boot Loader program. The Boot Loader program can use any available data interface and associated protocol to read code and write (program) that code into the Flash memory, or read the code from the Program memory. The program code within the Boot Loader section has the capability to write into the entire Flash, including the Boot Loader memory. The Boot Loader can thus even modify itself, and it can also erase itself from the code if the feature is not needed anymore. The size of the Boot Loader memory is configurable with Fuses and the Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

Features

• Read-While-Write Self-Programming

• Flexible Boot Memory size

• High Security (Separate Boot Lock Bits for a Flexible Protection)

• Separate Fuse to Select Reset Vector

• Optimized Page(1) Size

• Code Efficient Algorithm

• Efficient Read-Modify-Write Support

The Instruction set of AVR microcontroller is included in the Appendix.

4.3 PCI Interfacing Chip (MCS9835CV)

The MCS9835 is a PCI based dual-channel high performance UART with enhanced bi-directional parallel controller. The MCS9835 offers 16-Byte transmit and receive FIFOs for each UART channel and a 16-Byte FIFO for the printer channel. The MCS9835 performs serial-to-parallel conversions on data received from a peripheral device, and parallel-to-serial conversions on data received from its CPU. In addition, MCS9835 fully supports the existing Centronics printer interface as well as PS/2, EPP, and ECP modes.

The MCS9835 is ideally suited for PC applications, such as high speed COM ports and parallel ports. The MCS9835 is available in a 128-pin QFP package. It is fabricated using an advanced submicron CMOS process to achieve low drain power and high-speed requirements.

The MCS9835 can be used in both 3.3V and 5V PCI signaling environments. The MCS9835 is a pin-compatible replacement for the previous Nm9835. The Nm9835 is no longer offered. It was only warranted for use with a 5V power supply, and was only intended to operate in 5V PCI signaling environments. The MCS9835's new RoHS "Lead Free" package and 3.3V or 5V operation make it a much more flexible device that is better suited for new designs.

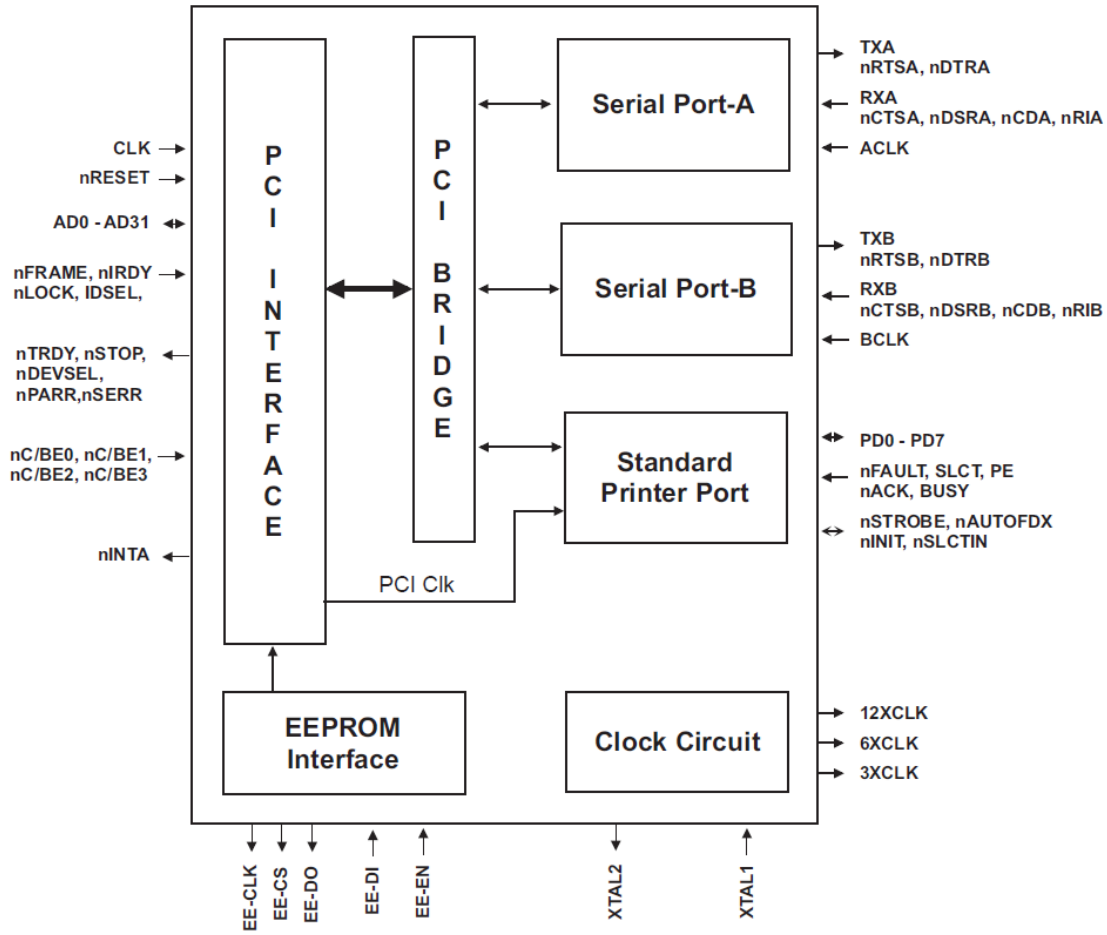


Figure 4.16 Block diagram of MCS9835CV

PCI Bus Operation:

The execution of PCI Bus transactions take place in broadly five stages: address phase; transaction claiming; data phase(s); final data transfer; and transaction completion.

Address Phase:

Every PCI transaction starts with an address phase, one PCI clock period in duration. During the address phase the initiator (also known as the current Bus Master) identifies the target device (via the address) and type of transaction (via the command). The initiator drives the 32-bit address onto the Address/Data Bus, and a 4-bit command onto the Command/Byte-Enable Bus. The initiator also asserts the /FRAME signal during the same clock cycle to indicate the presence of valid address and transaction information on those buses. The initiator supplies the starting address and command type for one PCI clock cycle. The target generates the subsequent sequential addresses

for burst transfers. The Address/Data Bus becomes the Data Bus, and the Command/Byte-Enable Bus becomes the Byte-Enable Bus for the remainder of the clock cycles in that transaction. The target latches the address and command type on the next rising edge of PCI clock, as do all other devices on that PCI bus. Each device then decodes the address and determines whether it is the intended target, and also decodes the command to determine the type of transaction.

Claiming the Transaction:

When a device determines that it is the target of a transaction, it claims the transaction by asserting /DEVSEL.

Data Phase(s):

The data phase of a transaction is the period during which a data object is transferred between the initiator and the target. The number of data Bytes to be transferred during a data phase is determined by the number of Command/Byte-Enable signals that are asserted by the initiator during the data phase. Each data phase is at least one PCI clock period in duration. Both initiator and target must indicate that they are ready to complete a data phase. If not, the data phase is extended by a wait state of one clock period in duration. The initiator and the target indicate this by asserting /IRDY and /TRDY respectively and the data transfer is completed at the rising edge of the next PCI clock.

Transaction Duration:

The initiator, as stated earlier, gives only the starting address during the address phase. It does not tell the number of data transfers in a burst transfer transaction. The target will automatically generate the addresses for subsequent Data Phase transfers. The initiator indicates the completion of a transaction by asserting /IRDY and de-asserting /FRAME during the last data transfer phase. The transaction does not actually complete until the target has also asserted the /TRDY signal and the last data transfer takes place. At this point the /TRDY and /DEVSEL are de-asserted by the target.

Transaction Completion:

When all of nIRDY, nTRDY, nDEVSEL, and nFRAME are in the inactive state (high state), the bus is in idle state. The bus is then ready to be claimed by another Bus Master.

PCI Resource Allocation

PCI devices do not have “Hard-Wired” assignments for memory or I/O Ports like ISA devices do. PCI devices use “Plug & Play” to obtain the required resources each time the system boots up. Each PCI device can request up to six resource allocations. These can be blocks of memory (RAM) or blocks of I/O Registers. The size of each resource block requested can also be specified, allowing great flexibility. Each of these resource blocks is accessed by means of a Base- Address-Register (BAR). As the name suggests, this is a pointer to the start of the resource. Individual registers are then addressed using relative offsets from the Base-Address-Register contents. The important thing to note is: plugging the same PCI card into different machines will not necessarily result in the same addresses being assigned to it. For this reason, software (drivers, etc.) must always obtain the specific addresses for the device from the PCI System. Each PCI device is assigned an entry in the PCI System’s shared “Configuration Space”. Every device is allocated 256 Bytes in the Configuration Space. The first 64 Bytes must follow the conventions of a standard PCI Configuration “Header”. There are several pieces of information the device must present in specific fields within the header to allow the PCI System to properly identify it. These include the Vendor-ID, Device-ID and Class-Code. These three fields should provide enough information to allow the PCI System to associate the correct software driver with the hardware device. Other fields can be used to provide additional information to further refine the needs and capabilities of the device. As part of the Enumeration process (discovery of which devices are present in the system) the Base- Address-Registers are configured for each device. The device tells the system how many registers (etc.) it requires, and the system maps that number into the system’s resource space, reserving them for exclusive use by that particular device. No guarantees are made that any two requests for resources will have any predictable relationship to each other. Each PCI System is free to use its own allocation strategy when managing resources.

For detail information the datasheet attached may be referred.

4.4 Microcontroller Programming Protocols – SPI and JTAG.

The microcontrollers have programmable flash memory and EEPROM memory that can be programmed using common techniques mentioned below:

Programming Interfaces

- In System Programming (ISP)
- High Voltage Serial Programming (HVSP)
- Parallel Programming (PP)
- JTAG Programming (JTAG Prog)
- PDI Programming on selected devices (PDI)

Debuging Interfaces

- JTAG (JTAG)
- debugWIRE (dW)
- PDI on selected devices (PDI)

All of the above mentioned have different physical and logical implementations. The Flash program memories, boot code memory, EEPROM memory, fuse bits, lock protection bits can only be programmed using the above techniques. The AVR microcontrollers have programmable flash memory and EEPROM that can be programmed while it is in the product using ISP technique. This is called In System Programming technique. Also they can also be programmed with the firmware running inside them, while the product is running is normal application. This is called In Application Programming technique.

For programming the AVR we require application software that will take the user input, one programming system to program the AVR with computer interface and the target AVR.

For example, take AVR Studio software for user input, AVR Dragon as programmer/debugger and ATmega128 as target microcontroller. At present, the PCI card is developed to be using JTAG interface for programming and debugging, detail discussion about JTAG physical and logical extension is given and short discussion about SPI programming is also described.

4.4.1 JTAG Interface

JTAG is commonly used to debug embedded systems and to program hardware devices. Companies like Atmel often provide JTAG interfaces on their products because of its popularity in industry. The full form of JTAG is Joint Test Action Group, the team of over 200 software companies, test and system vendor started in mid 80's. The standard was sanctioned by IEEE as Std. 1149.1 Test Access Port and Boundary Scan Architecture in 1990. The main focus of this standard was to build test facilities and testing points into chips and ensure compatibility among the compatible ICs.

Figure 4.16 shows a block diagram of the JTAG interface and the On-chip Debug system including TAP Controller. The TAP Controller is a 16 state machine controlled by the TCK and TMS signals and controls the operation of Boundary scan circuitry, JTAG programming circuitry, or On-chip Debug system. The TAP Controller selects either the JTAG Instruction Register or one of several Data Registers as the scan chain (Shift Register) between the TDI input and TDO output. The Instruction Register holds JTAG instructions controlling the behavior of a Data Register. The ID-Register, Bypass Register, and the Boundary-scan Chain are the Data Registers used for board-level testing. The JTAG Programming Interface (actually consisting of several physical and virtual Data Registers) is used for JTAG Serial Programming via the JTAG interface. The Internal Scan Chain and Break Point Scan Chain are used for On-chip Debugging only.

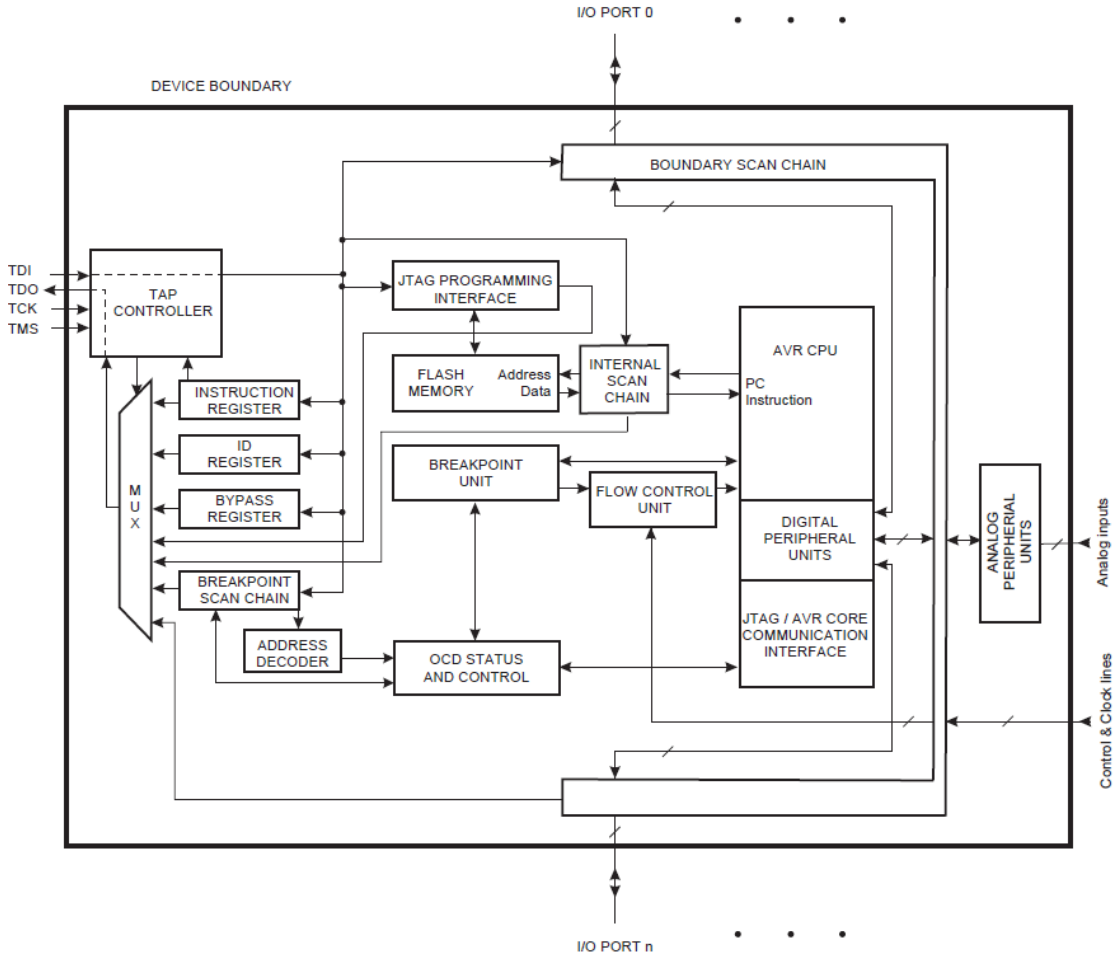


Figure 4.17 Block Diagram of JTAG peripheral in AVR(from datasheet).

The JTAG interface is accessed through four of the AVR's pins. In JTAG terminology, these pins constitute the Test Access Port – TAP. These pins are:

TMS: Test Mode Select. This pin is used for navigating through the TAP-controller state machine.

TCK: Test Clock. JTAG operation is synchronous to TCK.

TDI: Test Data In. Serial input data to be shifted in to the Instruction Register or Data Register (Scan Chains).

TDO: Test Data Out. Serial output data from Instruction register or Data Register.

The state transitions depicted in Figure 4.17 depend on the signal present on TMS (shown adjacent to each state transition) at the time of the rising edge at TCK. The initial state after a Power-On Reset is Test-Logic-Reset. The LSB is shifted in and out first for all Shift Registers.

Assuming Run-Test/Idle is the present state, a typical scenario for using the JTAG interface is:

1. At the TMS input, apply the sequence 1, 1, 0, 0 at the rising edges of TCK to enter the Shift Instruction Register – Shift-IR state. While in this state, shift the four bits of the JTAG instructions into the JTAG Instruction Register from the TDI input at the rising edge of TCK. The TMS input must be held low during input of the 3 LSBs in order to remain in the Shift-IR state. The MSB of the instruction is shifted in when this state is left by setting TMS high. While the instruction is shifted in from the TDI pin, the captured IR-state 0x01 is shifted out on the TDO pin. The JTAG Instruction selects a particular Data Register as path between TDI and TDO and controls the circuitry surrounding the selected Data Register.

2. Apply the TMS sequence 1, 1, 0 to re-enter the Run-Test/Idle state. The instruction is latched onto the parallel output from the Shift Register path in the Update-IR state. The Exit-IR, Pause-IR, and Exit2-IR states are only used for navigating the state machine.

3. At the TMS input, apply the sequence 1, 0, 0 at the rising edges of TCK to enter the Shift Data Register – Shift-DR state. While in this state, upload the selected Data Register (selected by the present JTAG instruction in the JTAG Instruction Register) from the TDI input at the rising edge of TCK. In order to remain in the Shift-DR state, the TMS input must be held low during input of all bits except the MSB. The MSB of the data is shifted in when this state is left by setting TMS high. While the Data Register is shifted in from the TDI pin, the parallel inputs to the Data Register captured in the Capture-DR state is shifted out on the TDO pin.

4. Apply the TMS sequence 1, 1, 0 to re-enter the Run-Test/Idle state. If the selected Data Register has a latched parallel-output, the latching takes place in the Update-DR state. The Exit-DR, Pause-DR, and Exit2-DR states are only used for navigating the state machine.

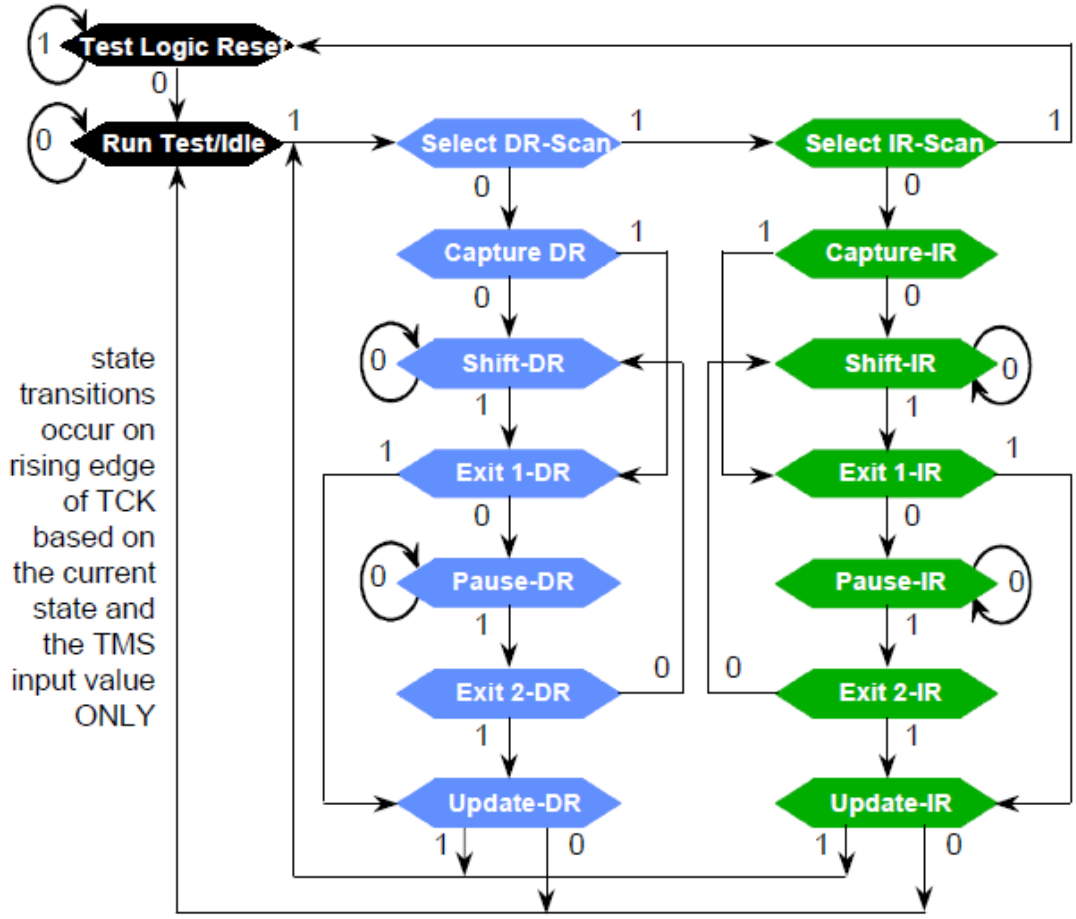


Figure 4.18 TAP Controller State Diagram

4.4.2 On-chip Debug System

As shown in Figure 4.18, the hardware support for On-chip Debugging consists mainly of:

• A scan chain on the interface between the internal AVR CPU and the internal peripheral units

• Break Point unit

• Communication interface between the CPU and JTAG system

All read or modify/write operations needed for implementing the Debugger are done by applying AVR instructions via the internal AVR CPU Scan Chain. The CPU sends the result to an I/O memory mapped location which is part of the communication interface between the CPU and the JTAG system.

The Break Point Unit implements Break on Change of Program Flow, Single Step Break, 2 Program Memory Break Points, and 2 combined Break Points. Together, the 4 Break Points can be configured as either:

• 4 single Program Memory Break Points

• 3 Single Program Memory Break Point + 1 single Data Memory Break Point

• 2 single Program Memory Break Points + 2 single Data Memory Break Points

• 2 single Program Memory Break Points + 1 Program Memory Break Point with mask (“range Break Point”)

• 2 single Program Memory Break Points + 1 Data Memory Break Point with mask (“range Break Point”)

The details of the OCD registers and JTAG instructions and related matter can be found from ATmega16 datasheet.

4.4.4 JTAG Programming.

Programming of AVR parts via JTAG is performed via the 4-pin JTAG port, TCK, TMS, TDI and TDO. These are the only pins that need to be controlled/observed to perform JTAG programming (in addition to power pins). It is not required to apply 12V externally. The JTAGEN Fuse must be programmed and the JTD bit in the MCUSR Register must be cleared to enable the JTAG Test Access Port.

The JTAG programming capability supports:

• Flash programming and verifying.

• EEPROM programming and verifying.

• Fuse programming and verifying.

• Lock bit programming and verifying.

The Lock bit security is exactly as in Parallel Programming mode. If the Lock bits LB1 or LB2 are programmed, the OCDEN Fuse cannot be programmed unless first doing a chip erase. This is a security feature that ensures no back-door exists for reading out the content of a secured device.

The details on programming through the JTAG interface and programming specific JTAG instructions are given in the section “Programming via the JTAG Interface” in the ATmega16L datasheet.

The standard header connection for JTAG signals is shown below:

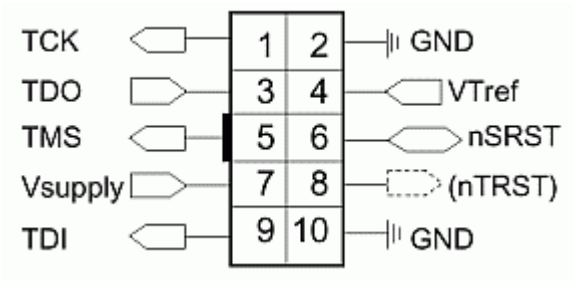


Figure 4.19 JTAG Header

JTAG Signals			
Pin	Signal	I/O	Description
1	TCK	Output	Test Clock, clock signal from JTAG ICE to target JTAG port
2	GND	-	Ground
3	TDO	Input	Test Data Output, data signal from target JTAG port to JTAG ICE
4	VTref	Input	Target reference voltage. VDD from target used to control logic-level onverter and target power LED indicator.
5	TMS	Output	Test Mode Select, mode select signal from JTAG ICE to target JTAG port
6	nSRST	Out-/In-put	Open collector output from adapter to the target system reset. This pin is also an input to the adapter so that a reset initiated on the target may be reported to the JTAG ICE.
7	Vsupply	Input	Supply voltage to the adapter, this connector can be used to supply the adapter with power from a regulated power supply(3 - 5)V DC (normally target VDD).This supply voltage input is automatically disconnected when a external power supply is connected
8	nTRST	NC(Output)	Not connected, reserved for compatibility with other equipment (JTAG port reset)
9	TDI	Output	Test Data Input, data signal from JTAG ICE to target JTAG port
10	GND	-	Ground

Table 4.9 JTAG Signals

4.5 SPI interface

The basic concept of the SPI interface can grabbed from figure 4.19. The figure shows that this interface uses minimum of three signal pins, and the data is transmitted and received serially, so sometimes it is also called 3 –

wire interface. This interface is originally developed by Motorola, but now it is standard serial interface for three wire communication. This same interface is used with maximum types of memory cards lik SD/MMC and different graphical LCD displays.

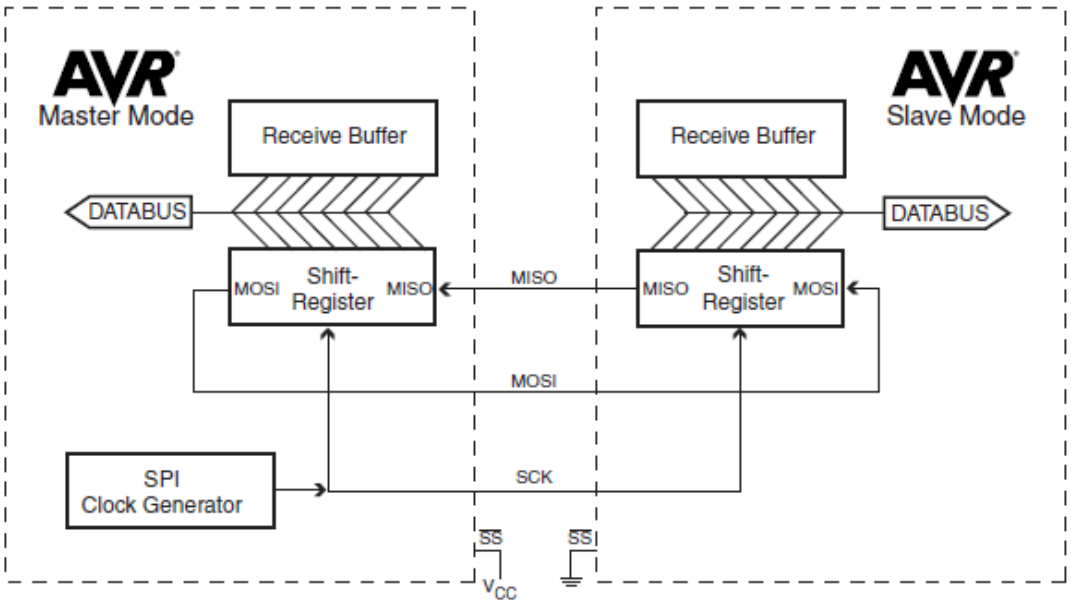


Figure 4.20 Master Slave interface for SPI

The SPI module allows a duplex, synchronous, serial communication between the MCU and peripheral devices. Software can poll the SPI status flags or the SPI operation can be interrupt driven.

The SPI includes these distinctive features:

- Master mode and slave mode
- Bi-directional mode
- Slave select output
- Mode fault error flag with CPU interrupt capability
- Double-buffered data register
- Serial clock with programmable polarity and phase
- Control of SPI operation during wait mode

The list of name and description of all pins including inputs and outputs that do, or may, connect off chip. The SPI module has a total of 4 external pins.

1 MOSI

This pin is used to transmit data out of the SPI module when it is configured as a Master and receive data when it is configured as Slave.

2 MISO

This pin is used to transmit data out of the SPI module when it is configured as a Slave and receive data when it is configured as Master.

3 /SS

This pin is used to output the select signal from the SPI module to another peripheral with which a data transfer is to take place when its configured as a Master and its used as an input to receive the slave select signal when the SPI is configured as Slave.

4 SCK

This pin is used to output the clock with respect to which the SPI transfers data or receive clock in case of Slave.

The SPI allows high-speed synchronous data transfer between the AVR and peripheral devices or between several AVR devices. On most parts the SPI has a second purpose where it is used for In System Programming (ISP). See application note AVR910 for details.

The interconnection between two SPI devices always happens between a master device and a slave device. Compared to some peripheral devices like sensors which can only run in slave mode, the SPI of the AVR can be configured for both master and slave mode. The mode the AVR is running in is specified by the settings of the master bit (MSTR) in the SPI control register (SPCR). Special considerations about the SS pin have to be taken into account. The master is the active part in this system and has to provide the clock signal a serial data transmission is based on. The slave is not capable of generating the clock signal and thus cannot get active on its own. The slave just sends and receives data if the master generates the necessary clock signal. The master however generates the clock signal only while sending data. That means that the master has to send data to the slave to read data from the slave. Data transmission between Master and Slave

The interaction between a master and a slave AVR is shown in Figure 4.19. Two identical SPI units are displayed. The left unit is configured as master while the right unit is configured as slave. The MISO, MOSI and SCK lines are connected with the corresponding lines of the other part. The mode in which a

part is running determines if they are input or output signal lines. Because a bit is shifted from the master to the slave and from the slave to the master simultaneously in one clock cycle both 8-bit shift registers can be considered as one 16-bit circular shift register. This means that after eight SCK clock pulses the data between master and slave will be exchanged. The system is single buffered in the transmit direction and double buffered in the receive direction.

This influences the data handling in the following ways:

1. New bytes to be sent cannot be written to the data register (SPDR) / shift register before the entire shift cycle is completed.

2. Received bytes are written to the Receive Buffer immediately after the transmission is completed.

3. The Receive Buffer has to be read before the next transmission is completed or data will be lost.

4. Reading the SPDR will return the data of the Receive Buffer. After a transfer is completed the SPI Interrupt Flag (SPIF) will be set in the SPI Status Register (SPSR). This will cause the corresponding interrupt to be executed if this interrupts and the global interrupts are enabled. Setting the SPI Interrupt Enable (SPIE) bit in the SPCR enables the interrupt of the SPI while setting the I bit in the SREG enables the global interrupts. The programming header connection is given in figure.

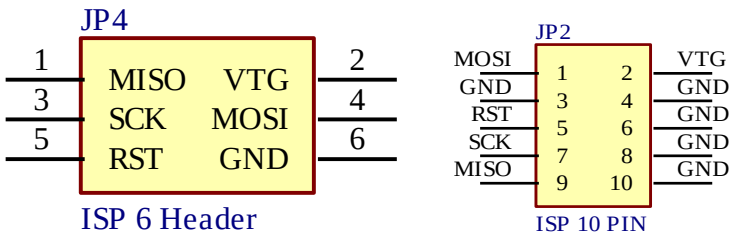


Figure 4.21 ISP 6 Pin headers

ISP Connector pinout				
Signal	**6-Pin**	**10-Pin**	**I/O**	**Description**
VTG	2	2	-	Power is delivered from the target board
GND	6	3,4,6, 8,10	-	Ground
MOSI	4	1	Output	Commands and data from AVRISP

				to target AVR
MISO	1	9	Input	Data from target AVR to AVRISP
SCK	3	7	Output	Serial Clock, Controlled by AVRISP
RESET	5	5	Output	Reset. Controlled by AVRISP

Table 4.10 ISP Signals

More on SPI interface and its block diagram will be found on the datasheet of ATmega128 or ATmega16L.

Chapter 5

Requirement Specification and Analysis

Chapter 5 Requirement Specification and Analysis

To procure the material for the research work at significant times, requirement specifications are obtained before starting development life cycle as mentioned in section 2.4 of chapter 2. On choosing the waterfall software development model as shown in figure 2.1 the concept phase and the requirement phase come into picture first. This phase is the most important phase in any development lifecycle, without requirement phase it is like starting to build a house without knowledge of architect design, place and government approval.

The recommended Laboratory equipments used for the research work are:

1. A reasonably feature-rich multimeter.
2. A good analog oscilloscope.
3. A laboratory power supply. It should have at least two independently adjustable DC current-limited outputs (30 V), with in-built current and voltage indicators.
4. A bench-mounted illuminated magnifier. This item is mandatory when working with surface-mounted parts, and it's useful even when working on larger packages.
5. A temperature-controlled soldering iron. Always keep a few spare tips on hand, also – especially working with surface-mount packages.

All of the above are available in Department of Electronics, Saurashtra University, Rajkot for research purposes.

5.1 Requirement Overview

The basic concept formed for the research work is shown in figure 5.1 that is self explanatory.

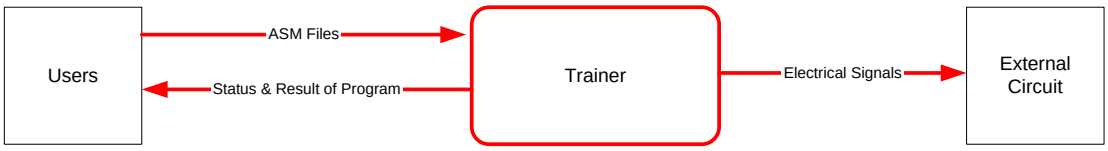


Figure 5.1 Basic Concept of Microcontroller trainer system

The brief concept is highlighted in figure 5.2 that shows the entities are required for research work.

Figure 5.2 Elaborated Concept

In the figure 5.2, the user has ability to develop assembler files in editor for the external circuit to work, the assembler converts it into hex files, these hex files are given to PCI card via the device driver software and the same are converted into electrical signals by PCI card, given to the external circuit. So, concluding the concept phase the requirement phase automatically comes into light. After doing thorough literature review as detailed in chapter 3, and accepting the discussions, five main entities coming in picture are:

- IDE (Integrated Development Environment)
- PCI Chip and its Device drivers.
- Interfacing Hardware.
- Selection of Microcontroller.
- Modules of Microcontroller System.

5.2 IDE (Integrated Development Environment)

AVR Studio is an Integrated Development Environment (IDE) for writing and debugging AVR applications in Windows 9x/ME/NT/2000/XP/VISTA/7 x32/64 environments. AVR Studio provides a project management tool, source file editor, simulator, assembler and front-end for C/C++, programming, emulation and on-chip debugging.

AVR Studio supports the complete range of ATMEL AVR tools and each release will always contain the latest updates for both the tools and support of new AVR devices.

AVR Studio 4 has a modular architecture which allows even more interaction with 3rd party software vendors. GUI plug-ins and other modules can be written and hooked to the system.

The purpose of the AVR SDK kit is to enable 3rd party developers as well as intern Atmel developers to write seamless extensions to AVR Studio 4. This includes the ability to (but not limited to):

- Quick generate a SDK project with the use of the SDK Wizard
- Extend the Graphical User Interface, including control with docking views, menus and toolbars.

- Access and process debug data.
- Access and process device specific data.
- Control 3rd party hardware.
- Integrate compilers.

Thus, AVR Studio is used as IDE and AVR Studio SDK is used for developing extension for AVR studio that will be described in section 7.2 of chapter 7

The AVR – GCC is an open source C complier and assembler that is part of GNU project and is available free for downloading. Collection of AVR software pre-compiled for Windows is

- C Compiler
- Assembler
- Linker
- Librarian
- File converter
    - Other file utilities
- C Library
- Programmer software
- Debugger
- In-Circuit Emulator software
- Editor / IDE
- Many support utilities

The AVR – GCC plugin turns AVR Studio into a GUI for AVR – GCC, which is automatically detected on installation of WinAVR distribution. To avoid hassles of command line install the WinAVR distribution after installing AVR Studio. This provides the total solution for assembling, compiling, simulating and debugging and programming the firmware for the target microcontroller. At present AVR toolchain installer is available that manages the plugin and WinAVR all together.

5.3 PCI Chips and its Device Drivers.

The PCI chip select is MCS9835CV which is a PCI with dual UART port. After installing the PCI card two UART ports for the PC will be available for working with. The interfacing hardware that includes the programmer and debugger is communicated via UART port. The exact control flow is described in section

6.2 of chapter 6. The MCS9835CV needs 22.1184MHz crystal and needs to be fabricated with 4 layers PCB.

The device drivers are obtained from the MosChip Semiconductor Technology LTD. for Windows and Linux OS versions. Installation procedure for the driver is described in section 9.2 of chapter 9.

5.4 Interfacing Hardware

The block that communicates between the software (IDE) and the external target (ATmega128) is known as the interfacing hardware. The interfacing hardware consist the hardware logic for programming the target, including handle communication between PC and target. This interfacing hardware consists of ATmega16L that is programmed with firmware to communicate with PCI chip MCS9835CV and target microcontroller ATnega128. The designing of hardware and firmware is described in chapter 7 and testing strategy is described in chapter 8.

5.5 Selection of microcontroller

Selection of target microcontroller is made after clarifying the points below:

- The device should be available for anonymous online or catalog ordering in single piece quantity from at least one major distributor. In the U.S., the big names commonly mentioned are Digi-Key, Newark, and Avnet Marshall. Digi-Key and Newark in particular have very broad inventories and generally allow purchases in small quantity. Avnet Marshall seems to cater more to manufacturing rather than prototype runs; they typically have 25- or even 250-piece minimum orders on parts. In India. Farnell and Dig-iKey have started the transaction in Indian rupees.
- Full data sheets for the device should be available without requiring a nondisclosure agreement or committing to any kind of purchase.
- A low-cost development board should be available for the part – either the manufacturer recommended board, a third-party board, or even some appliance based around the chip, as long as sufficient documentation exists to enable use of the appliance as a test bed for own code. Can ask the manufacturer and distributor if loaner boards

are available; if you can borrow a board for a month or two, it will be enough to get at least bootstrap code up and running and establish a basic level of familiarity with the microcontroller. Then move to your own hardware and return the evaluation board.

- There should be a direct technical contact available at the chip vendor, at least for emergency issues; it should not be necessary to route all questions through distribution. There are times when a complex problem will take weeks to solve when there are several layers in the communication chain, versus only a day or two if communicated directly with the *cognoscenti* at the chip manufacturer.
- The device should have been shipping to OEMs for at least three to six months.
- The core should be supported by the GNU toolchain.
- There should be at least one currently shipping commercial product that uses the device and the larger the market for this device, the better. All too often, parts that are consumed only by small niche markets are discontinued in favor of parts with more general applicability.

To be familiar with any microcontroller family the following steps must be taken into consideration:

1. Choose a microcontroller from the vendor's selection matrix.
2. Buy the vendor's evaluation board for this part.
3. Buy one of the commercial compilers and possibly a hardware debugging module recommended for the evaluation board.
4. License one of the operating systems recommended for the evaluation board.
5. Develop your application *in vitro* on the evaluation board.
6. Develop your hardware.
7. Port the operating system and your known-good application to the real hardware.

For quickly developing any application the approach mentioned can be very useful:

- Locate a third-party demonstration platform for the part of interest.

- Locate a consumer appliance based on the chip that interests you and reverse-engineer it enough to load your own firmware and patch on your own hardware.
- Design your own PCB and have it etched and populated either locally or (if this is a commercial project) by your factory; develop your firmware on this board while debugging the hardware at the same time.

Brooding on the above mentioned points the AVR 8 – Bit microcontroller is a better solution because of following features:

- High performance
  - True RISC architecture
  - Harvard architecture
  - True single cycle execution
  - 20 MIPS at 20Mhz
  - 32 general purpose registers
- Low power consumption
  - 1.8 - 5.5 volts operation
  - Sleep controller with a variety of operation modes
  - Fast wake-up from low-power modes
  - Software controlled operation frequency
  - Single cycle execution and high code density
- High code density
  - Architecture designed for C
  - 32 general registers
  - C-like addressing modes
  - 16- and 32-bit arithmetic support
  - Linear address maps
- Outstanding memory technology
  - 20 years of Flash memory experience
  - Self-programming Flash
  - EEPROM for parameter storage
  - Full integration on single die: Flash, EEPROM, SRAM
- High integration
  - Devices range from 1 to 256KB

- Pin count range from 8 to 100
- Full code compatibility
- Pin/feature compatible families
- One set of development tools

➢ In-System Development
- In-System Programming
- In-System Debugging
- In-System Verification

➢ Support
- Fully updated product web
- Highly skilled Field Application Engineers
- Support mail handled by AVR experts
- Reference designs
- Application notes
- AVRfreaks community website

The technical description of the AVR microcontroller is in section 4.2 of chapter 4.

5.6 Testing modules of Microcontroller System.

It is the basic requirement to be familiar with any microcontroller family, after selection of microcontroller the evaluation board or development board must be purchase with software license and work on it extensively publishing and presenting papers in journals and seminars. The online support and application notes must worked with that leads to accomplishment of the original research work. Having hands-on experience on all the peripherals of the microcontroller is important, and makes the entire concept clear for a newbie.

AVR STK – 500 and AVR Butterfly is used to learn the AVR 8 – Bit family of microcontrollers. AVR Dragon is used to debug and understand the serial interfaces like RS232, 1 – wire, I^2C, SPI, JTAG. Testing modules of microcontroller systems says about to be hands-on every technology related with the target microcontroller.

5.7 Discussion and summary

The literature survey of various PCI cards and tools, the theoretical background and successful completion of concept phase and requirement phase, the selection of chips and various hardware/ software tools can be finalized now. The table 5.1 shows the modular selection of entities for this research work according the section 5.1.

Basic modules	**Hardware**	**Software**	**Implementation**
IDE	Personal Computer with PCI card connector	AVR Studio, AVR toolchain	C Compiler, Assembler, Linker, Librarian, File converter, Other file utilities, C Library, Programmer software, Debugger, In-Circuit Emulator software, Editor, Many support utilities.
		AVR Studio SDK	Extend the features of AVR Studio.
PCI chip	MCS9835CV	Device Drivers	PCI Bus to UART
Interfacing HW	ATmega16L	Bootloader	To flash new firmware from PCI UART
		Firmware	JTAG programming and debugging
Target Micros	ATmega128	Test Sample	Demonstrate programming and debugging of target
Modules	LED, switch, 24C256 etc	Firmware from sample exercise	Demonstrate I^2C, SPI, I/O, UART, etc

Table 5.1 Modules of research work

Chapter 6

System Design and Test Plans

Chapter 6 System Design and Test Plan

According to the figure 2.1 Waterfall software development model, the concept phase and the requirement phase are completed in chapter 5. In this chapter, the high level design phase and the procedure of detail design phase will be described. The architectural design gives a broad idea about the process flow of the research work. The detail design breaks down architectural design into small subsystems to be worked on.

6.1 Architectural design.

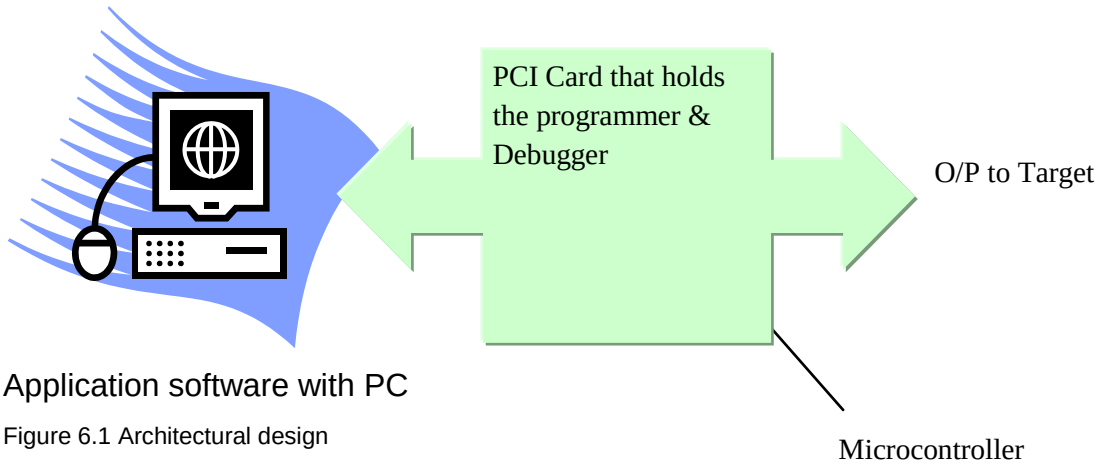


Figure 6.1 Architectural design

The figure 6.1 shows, that for the said research work application software running with the PC is needed and a PCI card that embeds the programmer and debugger for target microcontroller. The application software includes the AVR Studio IDE for AVR microcontroller and the device driver for PCI chip MCS9835CV. The PCI card includes the PCI chip itself and microcontroller chip that works as programmer and debugger.

6.2 Detail Design

The detail design phase exactly replicates the table 5.1 in chapter 5. Application software is bifurcated into two following parts:

- IDE – AVR Studio IDE, AVR Toolchain & AVR Studio SDK.
- Device Drivers – MCS9835CV device drivers.

The user setting of AVR Studio and extended development using AVR Studio SDK is discussed in section 7.2 of chapter 7. The device driver related implementation is also discussed herewith.

The detail block diagram of PCI card is shown in figure 6.2

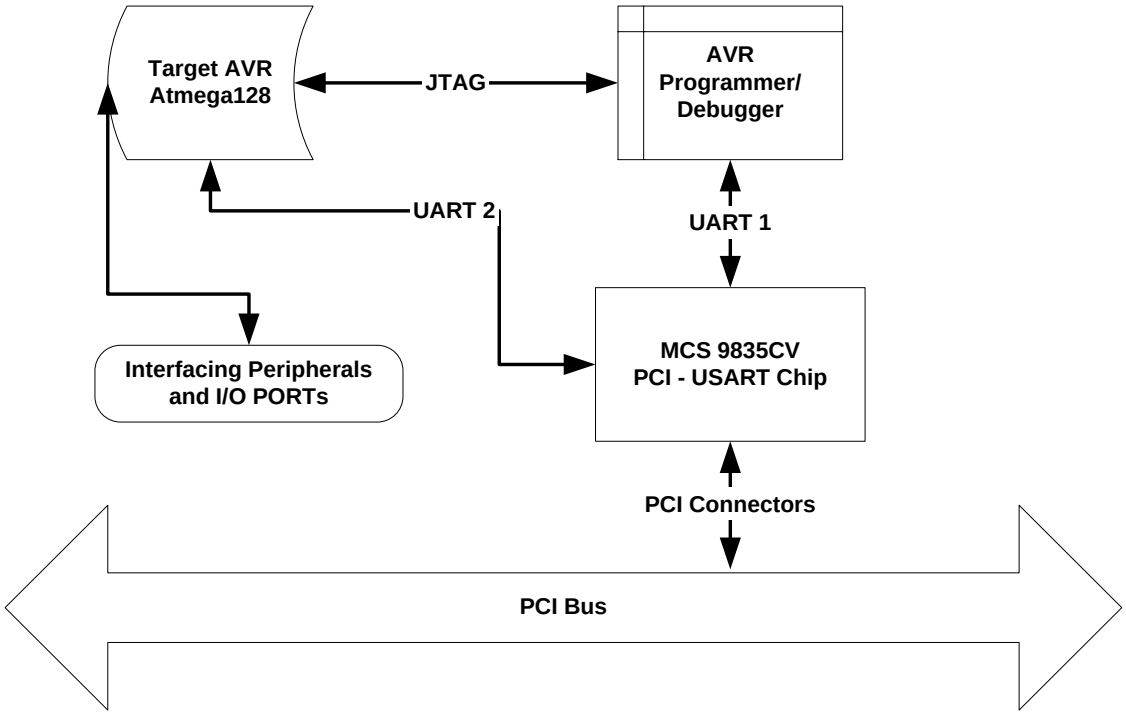


Figure 6.2 Detail block diagram of PCI card

The PCI card consists of following subsystems to be built up and tested:

- MCS9835CV PCI interfacing. The circuit design of the chip with PCI bus is described here and how it is associated with ATmega16L and ATmega128.
- ATmega16L based JTAG programmer and debugger. With the help of open source AVR-ICE project the JTAG port interfacing is tested and the JTAG-ICE is ported.
- Target ATmega128 prototype design. The interfacing onnections with peripherals are designed.
- Interfacing modules. Interfacing with UART of PCI chip with target, I/O port testing and EEPROM 24C256 read/write is done and other simple circuits.

A four layer PCB is developed to include all the subsystems listed above. The firmware, circuit and PCB design is discussed in chapter 7.

6.3 Test plan of detail design.

Proper test plan is to be implemented before designing the circuit in real. Improper way of testing and not following the development life cycle may lead

to wastage of time and money. Test plan verifies functioning of each subsystem the PCI card PCB with components mounted. The table 5.1 illustrates the modular subsystem as to what has to be tested before going to the next. Before attempting the design of 4 layered PCB all the sub-system must be individually designed programmed and test for proper functioning. The proper testing and implementation of each of them is in chapter 8. Simple test program and firmware may develop during the testing phase. All the sample exercise developed and tested here is listed in the appendix. The successful test result of all the subsystems leads to system integration and testing the PCI card which is working as JTAG programmer and debugger.

Chapter 7

System Development

Chapter 7 System Development

The requirement phase leads to system development, where each of the subsystem of the PCI card is to be designed and developed. After the design of all the subsystems the PCB for each of them is fabricated for final integration and deployment. The designing of circuit and the firmware along with PCB will be described together. This PCI card is developed so as to use it as debugger rather than programmer.

7.1 Individual Module Development

The step by step developing and taking the results of each step leads to next step. Special precautions are to be taken before working on the next subsystem, otherwise output from subsystems will not be fruitful to the next. Here development of plugin for the AVR Studio, about drivers of MCS9835CV, JTAG Programmer and Debugger, PCI interfacing, ATmega128 prototyping and its interfacing modules is done. The schematics and printed circuit board (PCB) are developed in Protel DXP 2004. The subsystems are developed in the following hierarchy.

1. Using the AVR Studio IDE and AVR Studio SDK.
2. MCS9835CV PCI device drivers.
3. MCS9835CV PCI Bus interfacing.
4. ATmega16L JTAG interfacing for debugger.
5. ATmega128 prototyping.
6. Interfacing Circuits.

After developing these individual subsystems, they will be implemented on a 4 layer PCB for final integration.

7.2 IDE and Drivers.

Choosing the ATmega128 as a target microcontroller for PCI card, the obvious choice for the Integrated Development Environment is AVR Studio from Atmel. The features of AVR Studio are discussed in section 5.2 of chapter 5. Alike all other IDE AVR Studio also has customizable text editor with color fonts, that are easy to differentiate the special keywords from the variables. AVR Studio operates in different modes, edit and debug mode. The tool added are only visible in the current working mode and when the tool item is added. Therefore, the tool is available two times in both edit and debug

mode. Also, the tools added when working in an assembler project will not be available when working with a C/C++ project.

In order to use any emulator/programmer using USB it is required to install the USB driver. Please do not connect the emulator/programmer to the computer before running the USB Setup. USB driver installation is done during the AVR Studio installation.

The AVR Studio supports both the assembler and the C complier for creating projects.

AVR Assembler

For small program systems it is not a bad choice, it supports all the different AVR parts through the it's DEVICE directive, which will help you from using instructions not supported in the device you have selected, and it is blistering fast due to the fact that it assembles executable code directly, there is no linker stage.

To get started is simple, when AVR Studio loads, select the AVR Assembler from the project dialog box, select a project name directory where the project shall reside, and click finish. A project file is created, an *.asm file is available in the editor window and you are set for writing your first instructions. Check out the online help. The AVR assembler has it's own book where all instructions and directives are explained. In addition, there is context sensitive help in the editor window, just write an instruction, place the cursor on top of the instruction and press F1, and you will get help on the syntax on the selected instruction.

If you develop for a specific device in mind you should include the *.def.inc file for the part. Each part has it's own *.inc file that defines all internal registers, bits and a lot of other stuff that makes it simpler for you to write code for the part. In addition the *.inc file sets the device directive for the assembler, letting the assembler know which part you are developing for.

The part files are found in the \ProgramFiles\Atmel \AVRTools\AVRAssembler \Àppnotes folder on your computer. A include file for ATmega128 will typically be named "m128def.inc". You do not have to give a path with the *.inc file as long as it is found in the default directory.

Press F7 in order to compile. The result of the compilation will show in the previously described Build view in the output window frame.

AVR C/C++ Compliers

There are a number of good C compilers available for the AVR family of microcontrollers including the compilers from IAR and Imagecraft which can be purchased, or the AVR GCC compiler that can be downloaded from a number of sites, including www.avrfreaks.com.

Using these compilers, a typical program in an environment is set up by the compiler vendor. You maintain your code project in a 3rd party tool. You typically set the compiler to produce code in DEBUG mode, which means that references to the high level code will be added to the compiled source. When you compile and link, you create an object file, and that object file can be loaded directly in AVR Studio 4. Different compilers use different object file formats. A number of formats are supported by the industry, some are proprietary for the compiler vendor company (like UBROF from IAR) and some are open and documented for everyone (like ELF used by the AVR GCC C compiler). In order to understand the code that your compiler and linker created AVR Studio must use an object file parser. The logical flowchart of creation of various files by AVR – GCC toolchain, WinAVR here is shown in figure.

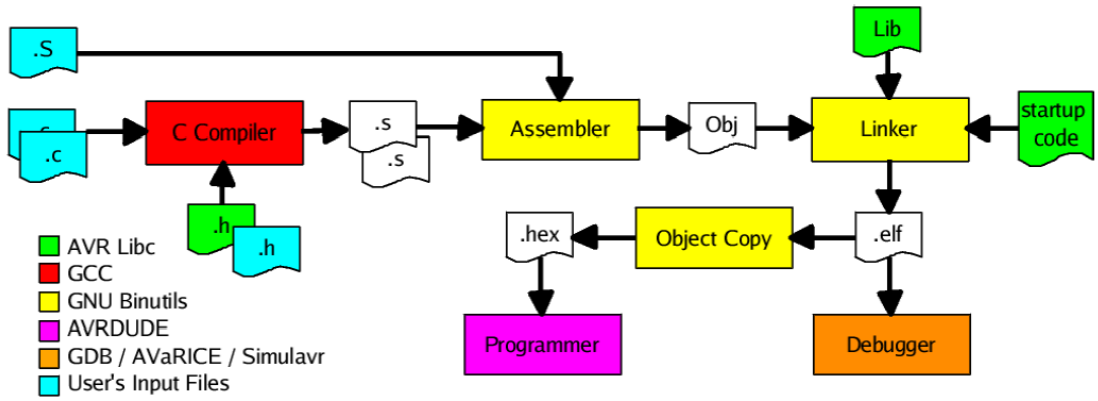


Figure 7.1 WinAVR(AVR – GCC Toolchain) Flowchart.

Currently AVR Studio 4 supports and will automatically detect UBROF (IAR), Nordic (AVR Assembler), COFF (Imagecraft, Codevision, ELAB, etc), and Intel Hex object files. An ELF/DWARF object file parser is in the planning.

In order to use an externally created object file with the AVR studio, use the Open File option in the File pulldown menu. If the file is created in debug mode, AVR Studio will create some magic: It opens the file, detects which

object file format is used, locks up the source code and displays the source code in the editor view. If you have an emulation platform available (or have selected the internal simulator) you are ready to rock: Pressing F5 will start the selected debug mode. You can set breakpoints in the code view, you can drag-and-drop variables down to the watch view, and you can manipulate the controller through the use of the I/O view. This is used for debugging the programs developed in BASCOM-AVR compiler.

The latest version of AVR Studio can be downloaded from www.atmel.com after free registration. The detail usage will be described in the AVR Studio Tool HTML help file.

AVR Studio 4 SDK(Software Development Kit)

The purpose of the kit is to enable 3rd party developers as well as intern Atmel developers to write seamless extensions to AVR Studio 4. This includes the ability to (but not limited to):

- Quick generate a SDK project with the use of the SDK Wizard
- Extend the Graphical User Interface, including control with docking views, menus and toolbars.
- Access and process debug data.
- Access and process device specific data.
- Control 3rd party hardware.
- Integrate compilers.

In addition AVR Studio 4.12 is enhanced with a plug-in manager. The user controls which plug-ins to start (and which to not start) through the use of the plug-in manager, before AVR Studio 4 is started. The Plugin-Manager is shown in figure 7.2. it can be found in Start menu item as shown in figure .

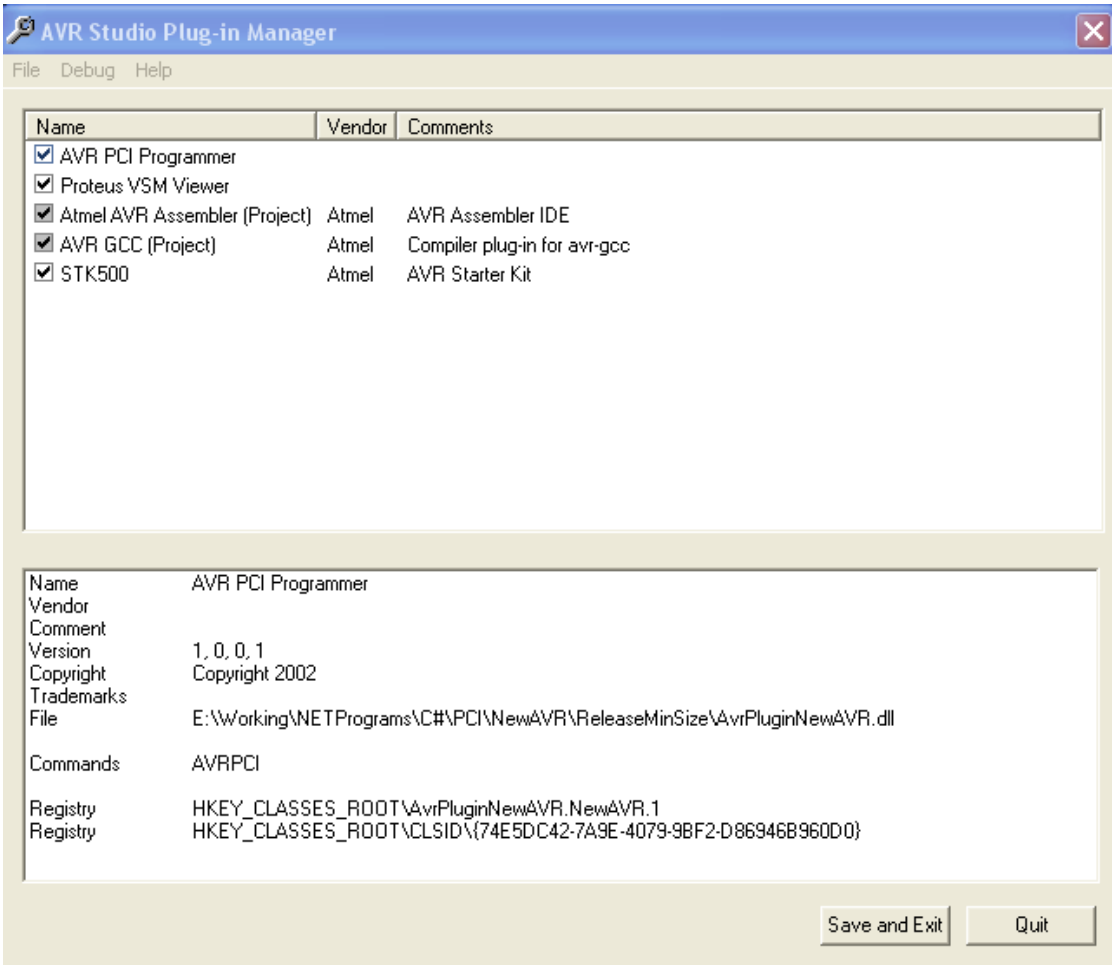


Figure 7.2 Plugin-Manager supporting AVR Studio.

Writing plug-ins for AVR Studio 4 requires skills in using:

- AVR Studio 4.
- Microsoft Visual Studio Integrated Development Environment.
- Microsoft Distributed Common Object Model (DCOM) technology.
- Deployment and installation technology (Microsoft Installer, Install Shield..)

Two types of plug-ins can be developed; GUI plug-in and project plug-in.

GUI plug-in

Add multiple numbers of commands, toolbars, buttons, menu items and accelerator keys to the main application. Automatically support full user customization. Add and display multiple number of docking windows. Get full access to the system services and target drivers. A GUI plug-in can be a dialog based window, a docking window or part of a multiple tabbed docking window.

Project plug-in

Support all functionality of a GUI plug-in, but have also the special hooks to support all the different project types. Only one instance of a project type plug-in can be open at the same time. For example, an assembler project cannot be open simultaneously with a compiler project. All the specific GUI elements are activated at once the project are created or opened. When the project are closed, all the specific GUI elements are removed and automatically saved for proper restore the next time the project are reopened.

AVR Studio 4 is built around the Microsoft DCOM technology. The Graphical User Interface (GUI) is the only executable in the system; all other functionality is provided through DCOM objects that are started either by the GUI executable, or by other DCOM objects.

AVR Studio 4 is a Single Process - Multi Threaded application. It means that all the separate threads lives within the Single Process address space and dies when the Single Process dies. It is the GUI that defines the Single Process, thus all DCOM objects in the system dies when the GUI executable dies.

Developing GUI plugin.

Here a GUI plugin is developed using this SDK. The SDK Plug-in Wizard is used in order to create skeleton code for an AVR Studio 4 SDK project. It is installed as a part of the SDK, and it is available from the Start menu in the AVR Tools folder.

Step 1

The SDK Plug-in Wizard is used in order to create skeleton code for an AVR Studio 4 SDK project. It is installed as a part of the SDK, and it is available from the Start menu in the AVR Tools folder.

Figure 7.3 Plugin Wizard for AVR Studio SDK.

Step 2:

After having pressed the Finish button the Wizard is ready to create the GUI plug-in project and asks you to confirm that you want to create the project. Click Create to create the project. The Wizard has now created a Visual Studio project for the plug-in in the folder specified as shown in figure 7.4.

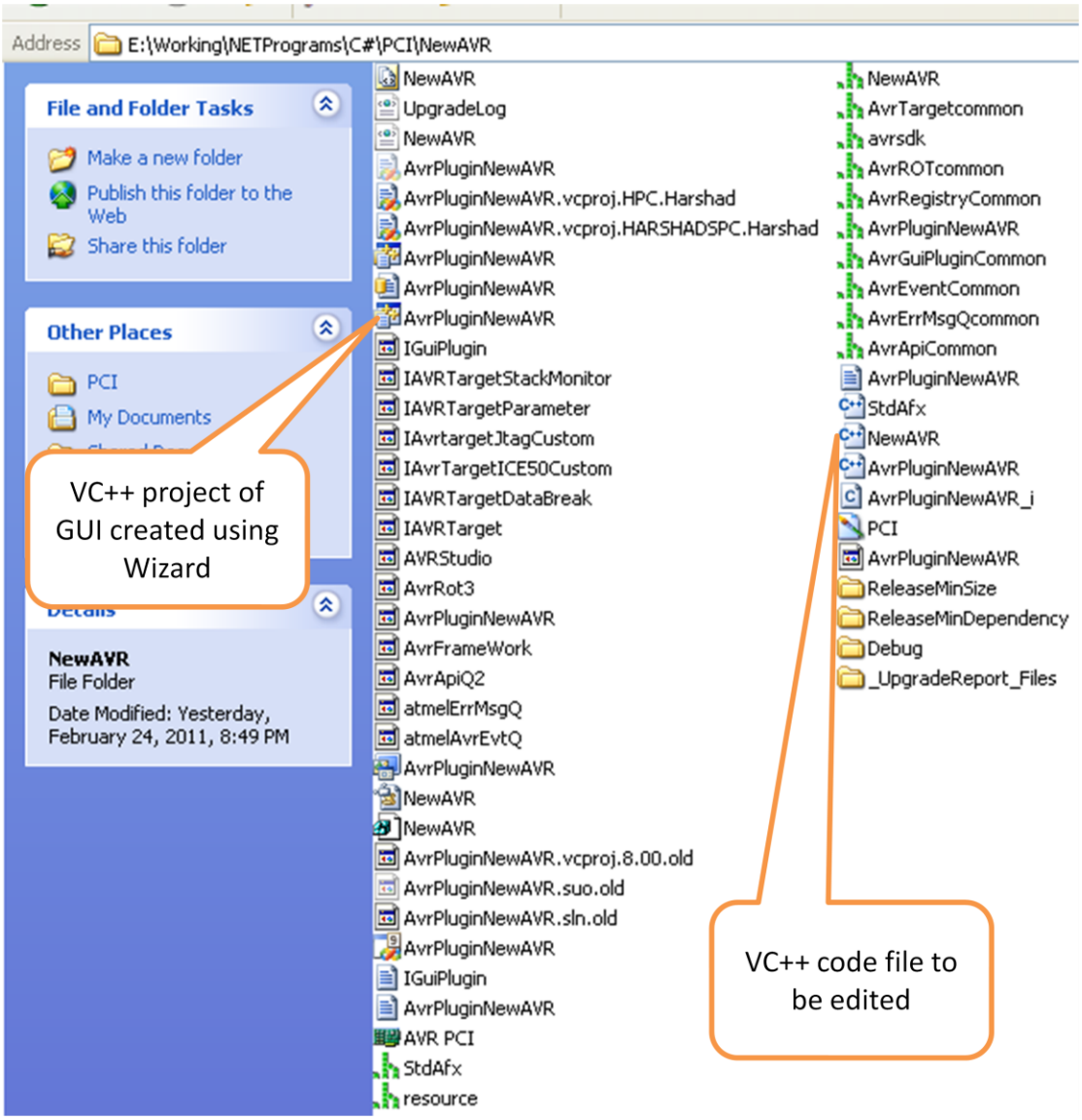


Figure 7.4 Folder of new created plugin.

The VC ++ project file shown in the figure is created using Plugin wizard, this has to be imported in the Visual Studio 2008. Here VC++ code file must be edited for having customized features. In the project properties, include the SDK files before compilation of the project. The coding for this project is included in the CD-ROM.

```
"<GUI_PLUGIN>"

   "<FRAMEWORK>"
      "<NAME>AVR PCI Programmer</NAME>"
      "<COMMAND_LIST>[AVRPCI]</COMMAND_LIST>"
```

```
"</FRAMEWORK>"

"<MENUS>"
   "<MENU_NAMES>[TOOLS]</MENU_NAMES>"
   "<TOOLS>"
   "<MENU_TEXT>Hari Prog</MENU_TEXT>"
      "<SUB_MENU>TRUE</SUB_MENU>"
      "<MENU_NAMES>[AVRPCI]</MENU_NAMES>"
      "<AVRPCI>"
         "<MENU_TEXT>AVR PCI Prog</MENU_TEXT>"
         "<SUB_MENU>FALSE</SUB_MENU>"
         "<TOOLTIP_HELP>Show NewAVR</TOOLTIP_HELP>"
         "<COMMAND_ID>AVRPCI</COMMAND_ID>"
         "<BITMAPID>196</BITMAPID>"
      "</AVRPCI>"
   "</TOOLS>"
"</MENUS>"

"<TOOLBARS>"
   "<TOOLBAR_NAMES>[AVRPCI]</TOOLBAR_NAMES>"
   "<AVRPCI>"
      "<COMMAND_LIST>[AVRPCI]</COMMAND_LIST>"
      "<AVRPCI>"
         "<COMMAND_ID>AVRPCI</COMMAND_ID>"
         "<TOOLTIP_HELP>AVR PCI Programmer</TOOLTIP_HELP>"
         "<BITMAPID>196</BITMAPID>"
      "</AVRPCI>"
   "</AVRPCI>"
   "</TOOLBARS>"

   "</GUI_PLUGIN>";
```

Looking at the above XML coding, the MENU tag is used to populate the menus and Toolbar tag is used to load the button in the toolbar.

The following code is written for execution of an external program Calculator on the press of the button PCI.

```
HRESULT CNewAVR::ExecuteCommand(LONG command)
{
      AFX_MANAGE_STATE(AfxGetStaticModuleState())
//commands from their index order are executed
```

```
	switch (command)
	{//programming done
		case 0:
		//run executable file
		HINSTANCE qq=ShellExecute(NULL, "open","calc.exe", NULL,
NULL, SW_SHOWNORMAL);
		 if((int)qq>32)
		   HRESULT         hr         =         m_iAvrFrameWork-
>errMsgWrite(EMSG_INFORMATION,CComBSTR("AVR         PCI:"),CComBSTR("
"),CComBSTR("AVR PCI Programmer Open"));
		    if((int)qq==2)
		    HRESULT         hr         =         m_iAvrFrameWork-
>errMsgWrite(EMSG_ERROR,CComBSTR("AVR               PCI:"),CComBSTR("
"),CComBSTR("AVR PCI Programmer executable not found"));
		 //   if(IDOK == AfxMessageBox("AVR PCI Programmer \n by
Chirutkar Harshad",MB_OK))
		//               HRESULT     hr     =     m_iAvrFrameWork-
>errMsgWrite(EMSG_INFORMATION,CComBSTR("AVR         PCI:"),CComBSTR("
"),CComBSTR("AVR PCI Programmer Close"));
 break;
	}
	return S_OK;
}
```

Step 3:

The project is built with Microsoft Visual Studio C++ 2008.NET. The DLL file generated is AvrPluginNewAVR.dll, which is then registered using command regsvr32.exe from Run program of Start menu.

Step 4:

Start AVR Studio. The new plug-in will display an empty button and an menu entry under Tools as shown in figure7.5. Also start the plugin manager to check the loaded plugin as shown in figure 7.2.

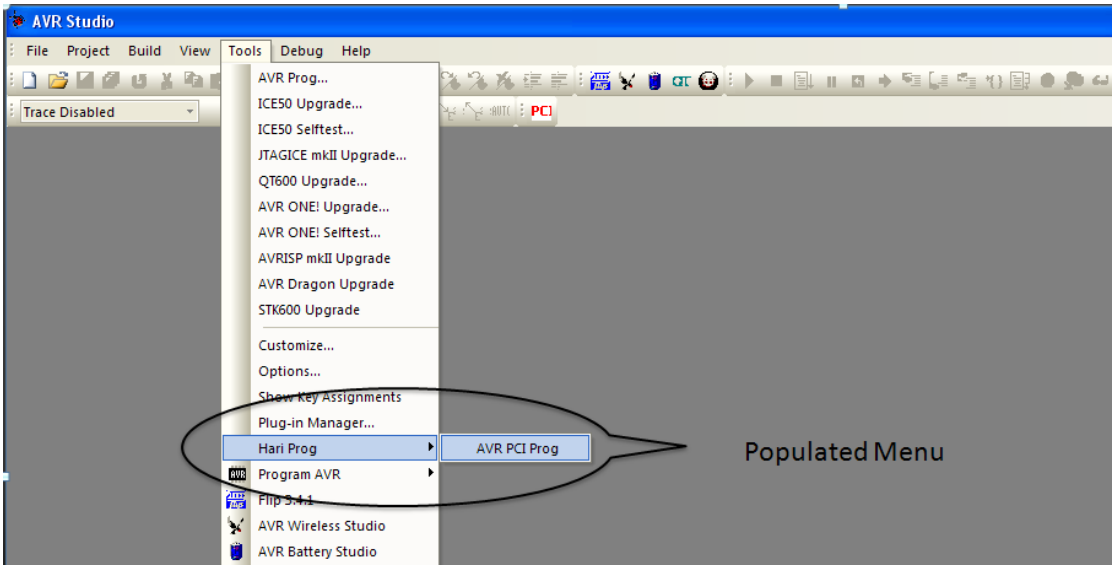


Figure 7.5 GUI Tool added in Tools Menu.

To execute the GUI tool click on AVR PCI Prog as shown in figure 7.5 and the executed example of calculator is shown in figure 7.6.

The above DLL file is generated from the examples given by the AVR Studio SDK. SDK is not free or open source software. The software of SDK was received after signing the license agreement between the Department of Electronics, Saurashtra University, Rajkot and Atmel Corporation.

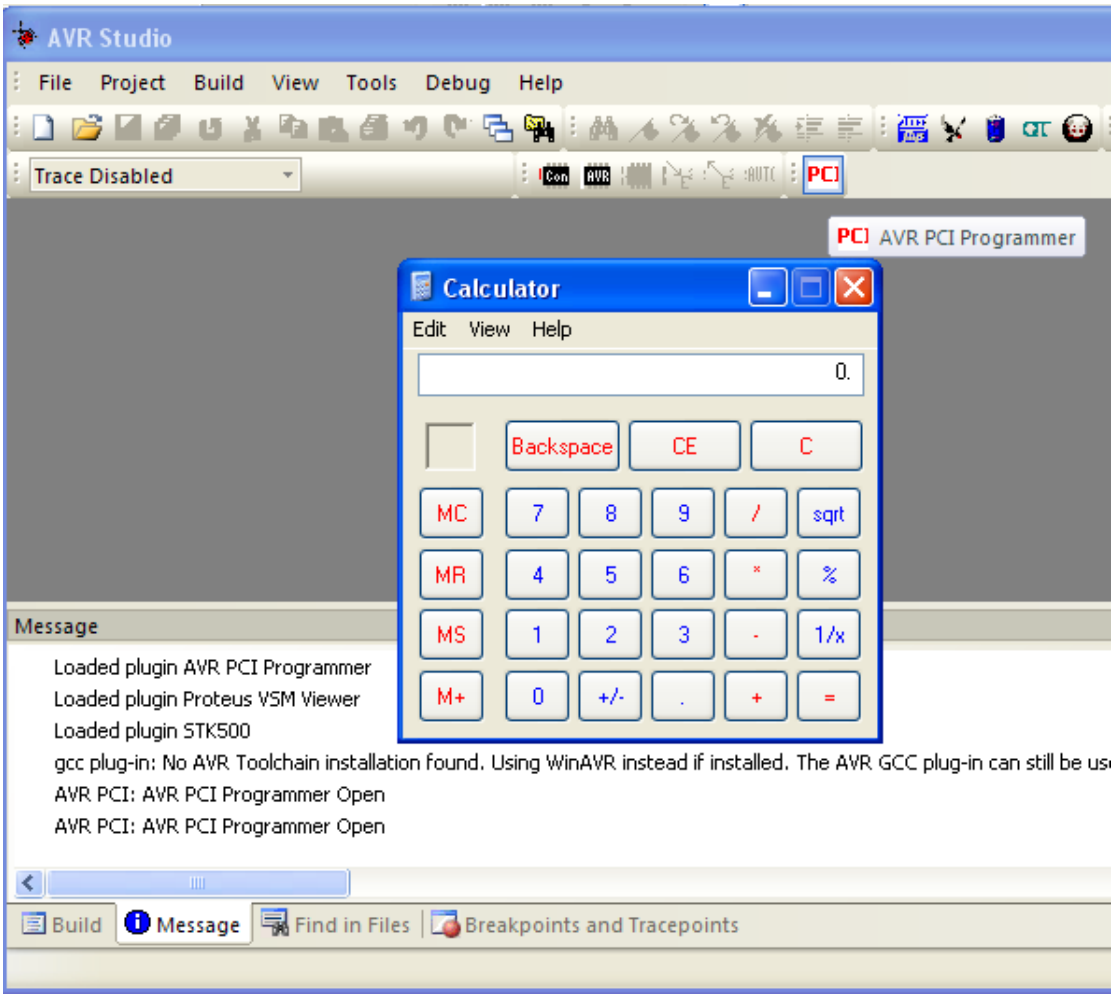


Figure 7.6 GUI tool executed.

MCS9835CV Device Drivers

The device drivers are provided by MosChip Technology. The hardware software requirements are in section 9.1 of chapter 9. The installation procedure for installing drivers for MCS9835CV is given in section 9.2 of chapter 9. The drivers are certified by the Windows Hardware Quality Lab (WHQL). After installations by default two UART and one Parallel port is available with the PC.

One of the UART is used to communicate with the ATmega16L that acts as a debugger and the other UART is left for communication with target ATmega128. The parallel port is not used.

MCS9835CV PCI Bus interfacing circuit design.

The details regarding the PCI chip is discussed in section 4.3 of chapter 4. The PCI edge connector has 124 pins in total, 62 pins on both sides, where

only 50 pins are used for interfacing with MCS9835CV. The figure 7.7 shows the PCI card edge schematics designed for 5V, 7.5W operation.

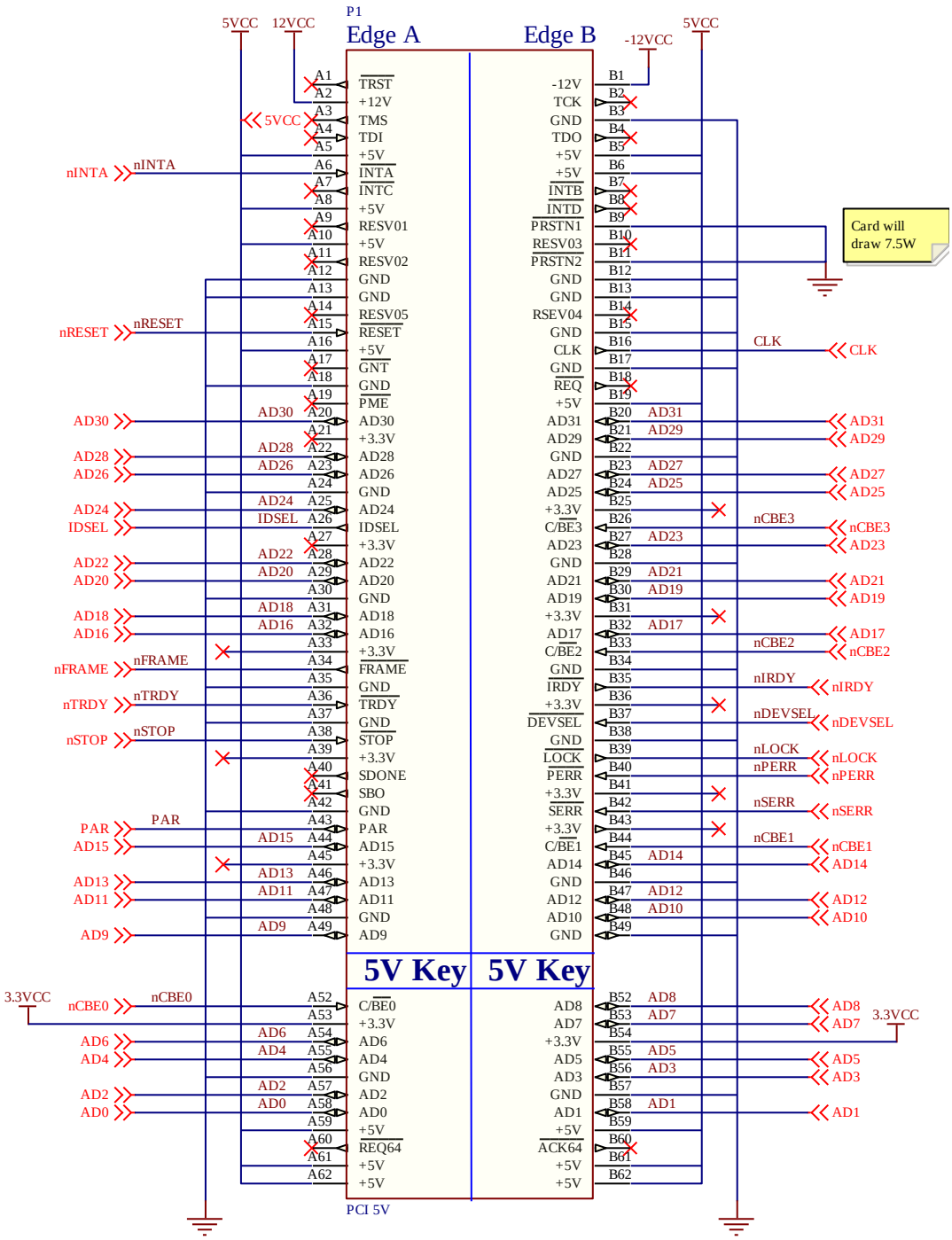


Figure 7.7 PCI 5V Edge Connector.

The pin-outs are arranged so that every signal pin is adjacent to a power or ground rail, which helps to reduce electromagnetic interference (EMI) by capacitive decoupling. The compact size is obtained by multiplexing the 32

address and data lines so they share the same 32 pins. This allows PCI cards to be made short enough to install in portable PCs.

Pair of pins B9 and B11 on the bus connector allows the system to determine the power requirements of the installed hardware. Interpreted as two bits they permit a total of four combinations showing that the slot is either empty, or contains a board with a power consumption of up to 7.5W, 15W or 25W.

The interfacing pins from MCS9835VC to PCI card is shown in figure 7.7.

The figure 7.8 shows that the external EEPROM is disabled by connecting Pin 123 EE-EN to ground. When the EEPROM is disabled, default values for PCI configuration registers will be used. Crystal oscillator input or external clock input pin (22.1184 MHz) is at pin 62 XTAL1. This signal input is used in conjunction with XTAL2 to form a feedback circuit for the internal timing. Two external capacitors (10pF) connected from each side of the XTAL1 and XTAL2 to GND are required to form a crystal oscillator circuit.

The UART-A is used to communicate with ATmega16L.The Pins RXA and TXA are connected to TXD and RXD pins of ATmega16L respectively. The UART-B port is used to communicate with target ATmega128. The Pins RXB and TXB are connected to TXD and RXD pins of ATmega128 respectively.

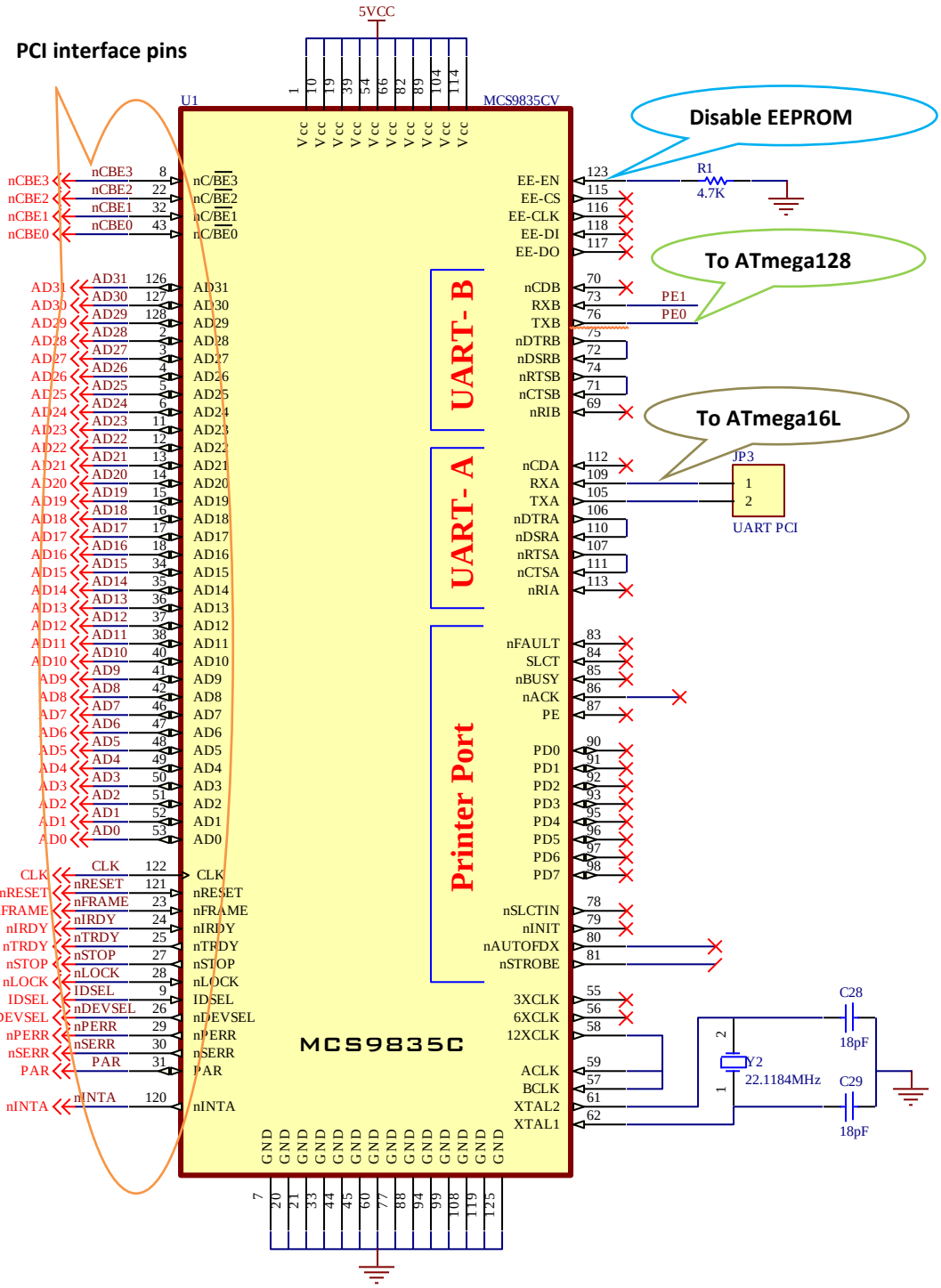


Figure 7.8 MCS9835CV connection schematic.

The Pin 58 12XCLK is the master clock out divided by the crystal frequency reduced to 1.8432MHz, gives standard UART clock for 155.2K data rate is connected to pins 59 ACLK and 57 BCLK. This provides standard clock to both the UARTs. The header JP3 is used to testing purpose only and communicates with the ATmega16L via UART interface.

MCS9835CV PCI interface PCB design.

System operation frequencies are increasing, PCB layout is becoming increasingly complex. A successful high-speed PCB must effectively integrate the IC's and other components into a single design. As Fast edge rates can contribute to noise generation, signal reflection, cross-talk and ground bounce; designers must be careful to handle these issues during PCB layout.

Besides the interlacing of power, ground and signal traces, PCI uses another innovation, reflected wave switching, to reduce power consumption and the EMI problem associated with fast, high power digital electronics. Circuit traces on a PCI board are unterminated. This means that a signal travelling along a trace meets high impedance at the end, and is consequently reflected back along the trace instead of being absorbed. By careful design, the logic gates are placed at the points where the incident and reflected waves reinforce each other. Because the voltages of the two waves add, the logic drivers need only produce a signal of half the needed voltage level, which reduces the power needed by a similar fraction. The PCI edge connectors are gold plated including the whole PCB for optimum performance.

PCI layout and routing guidelines for the PCI card is listed below.

1. Place the 22.1184MHz Crystal close to the MCS9835CV chip. Place these two nearer to the PCI Connector on the board. Route the clock signals over continuous ground and power planes. Shielding or GND Plane should be provided for these clock Signals on layers.

2. The maximum trace lengths for all 32-bit interface signals are limited to 1.5 inches for 32-bit and 64-bit add-in cards. This includes all signal groups except those listed as "Systems pins," Interrupt pins", "SMBus" and "JTAG pins."

3. The trace lengths of the additional signals used in the 64-bit extension are limited to 2 inches on all 64-bit add-in cards.

4. Trace lengths for the PCI CLK signal is 2.5 inches ± 0.1 inches for 32-bit and 64-bit add-in cards and must be routed to only one load.

5.. When it becomes necessary to turn 90°, use two 45° turns or an arc instead of making a single 90° turn. This reduces reflections on the signal by minimizing impedance discontinuities.

6. Do not route signals under crystal oscillator, clock-synthesizers, magnetic devices or ICs that use and/or duplicate clocks.

7. Route high-speed signals over continuous ground and power planes.

8. Provide ample power and ground planes

The part of the routed PCB showing MCS9835CV is shown in figure.

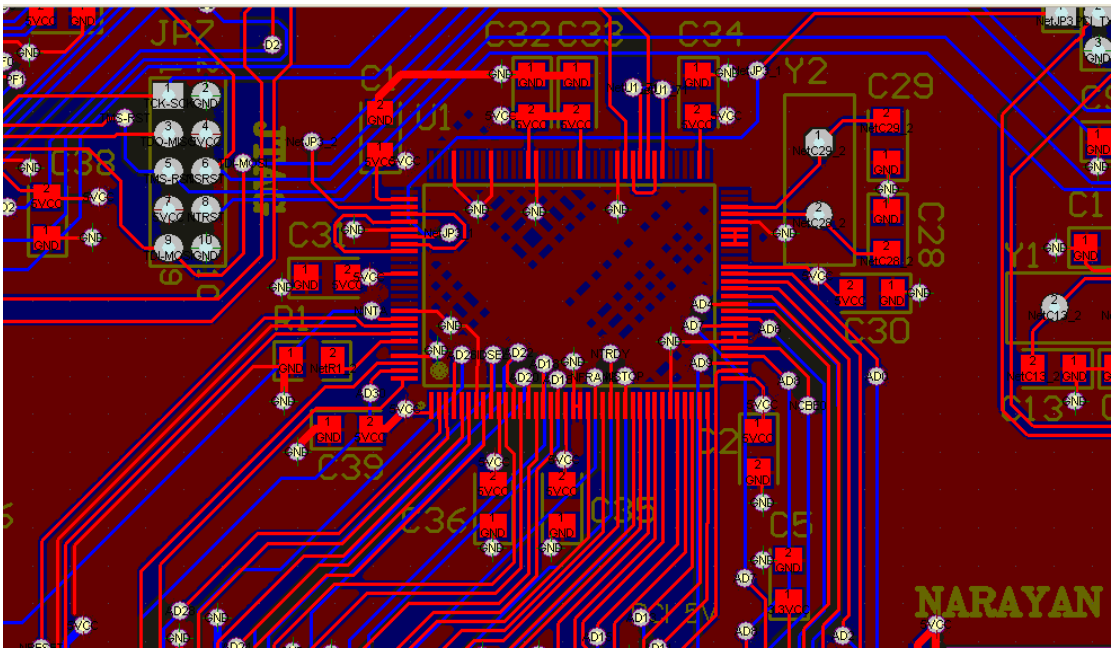


Figure 7.9 PCB pattern of MCS9835CV

Layer Stack Details of the PCI card is shown in figure 7.10.

1. Top Layer.
2. GND Layer.
3. VCC Layer.
4. Bottom Layer.

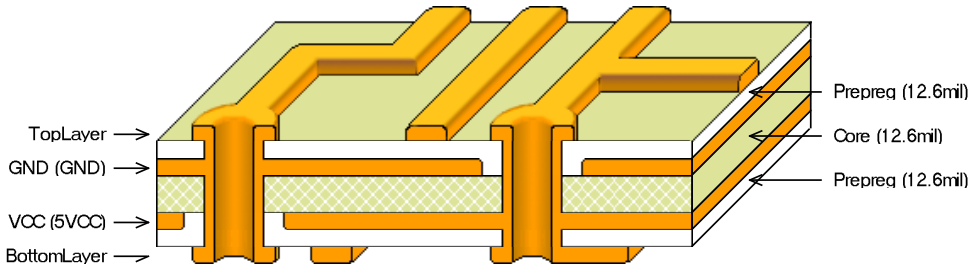


Figure 7.10 Stack Layer details.

The PCB net route length is given in the appendix.

7.3 JTAG Programmer/Debugger firmware (PCB).

Circuit Design

The circuit design of JTAG Programmer/debugger is inspired from the old JTAG-ICE (JTAG-In circuit Emulator) tool available from the Atmel. Not all features are adopted from the reference circuit. So the developed circuit with firmware works and recognized as JTAG-ICE debugger. The circuit for ATmega16L is shown in figure 7.11.

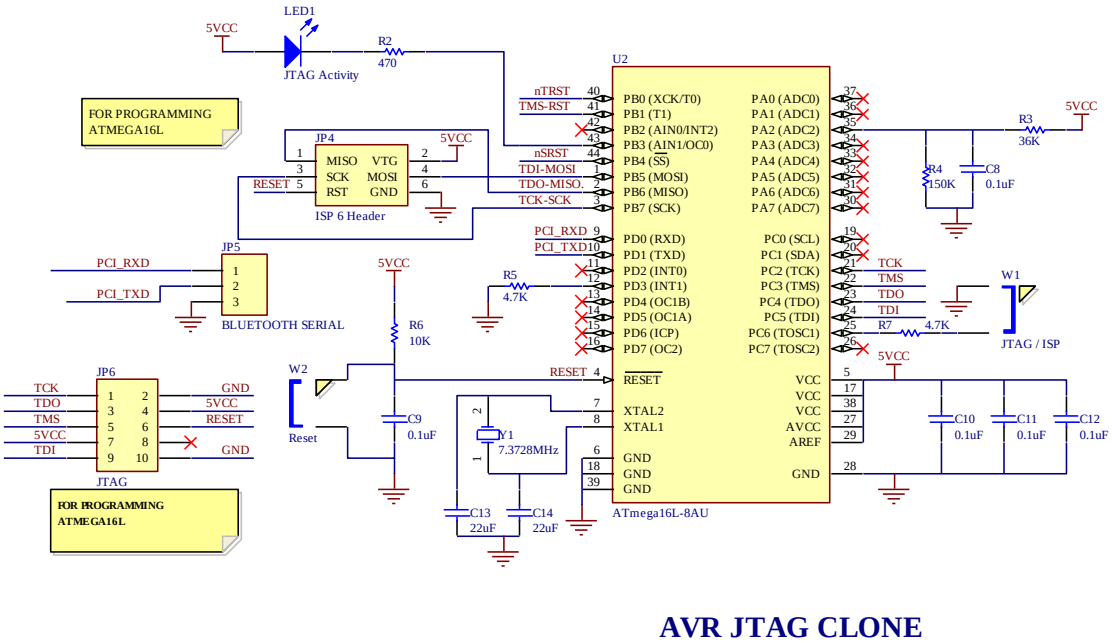


Figure 7.11 AVR ATmega16L JTAG Circuit.

The ATmega16L is used, as a replacement of its obsolete counterpart. It is the 44 pin QFP surface mount package. The heart of any microcontroller system is its clocking circuit; here 7.3728 MHz quartz crystal is as a clock. XTAL1 and XTAL2 are input and output, respectively, of an inverting amplifier which can be configured for use as an On-chip Oscillator. Such an odd value of crystal is taken so as to have minimum error for data rate of UART serial communication and is suitable for low power version of ATmega16L.

To communicate with the ATmega16L there are three headers JP4 is 6-pin for ISP serial programming, JP5 is 2-pin for UART serial communication and JP6 is 10-pin for JTAG programming. The header JP7 is 10 pin used to program the target ATmega128. The example of ISP and JTAG headers are given in figure 4.20 and 4.19 respectively in chapter 4.

The JP5 header is used here for testing purpose only. It is also used to communicate with JP3 of MCS9835CV. Here transmit and receive pins must be crossed for having communication between ATmega16L and PCI chip.

The bootloader is designed in such a way that pin 25 PC6 of ATmega16L must be held low for ISP programming the firmware, so there is a two pin header W1. The pin 25 PA2 is used by the JTAG-ICE[23] for measuring the target voltage VTG. The pin 12 PD3 is pulled down so as to tell the firmware that target power is always correct. The following table 7.1 gives the description of the ATmega16L pins connected for the JTAG header JP7. Actually there are direct nets from ATmega16L to target but JP7 is put on the board to use target from external JTAG programmer/debugger.

Pin	Signal	I/O at JP7	Description
1	TCK-SCK	Output	Test Clock, clock signal from ATmega16L JTAG ICE to target JTAG port
2	GND	-	Ground
3	TDO-MISO	Input	Test Data Output, data signal from target JTAG port to ATmega16L JTAG ICE
4	VTref(5VCC)	Input	Target reference voltage. 5VCC from target used to control logic-level converter
5	TMS-RST	Output	Test Mode Select, mode select signal from ATmega16L JTAG ICE to target JTAG port
6	nSRST	Out-/In-put	Open collector output from adapter to the target system reset. The pin 44 PB4 is also an input to the adapter so that a reset initiated on the target may be reported to the JTAG ICE.
7	Vsupply (5VCC)	Input	Supply voltage to the adapter, this connector can be used to supply the adapter with power from a regulated power supply(3 - 5)V DC (normally target 5VCC).
8	nTRST	NC(Out put)	Not connected, reserved for compatibility with other equipment (JTAG port reset)

9	TDI-MOSI	Output	Test Data Input, data signal from ATmega16L JTAG ICE to target JTAG port
10	GND	-	Ground

Table 7.1 Pin signals of JP7 in figure 7.11

The pin 43 PB3 sinks small amount of JTAG LED current, so as to show the activity of the ATmega16L. the firmware for bootloader can be programmed via ISP header JP4 by SPI serial programming or via JTAG header JP6 by external JTAG programmer.

PCB Design

The PCB design for the ATmega16L circuit is simple, and the normal net routing protocols can be followed for designing the PCB layout. The figure 7.12 shows the ATmega16L part on the routed PCB.

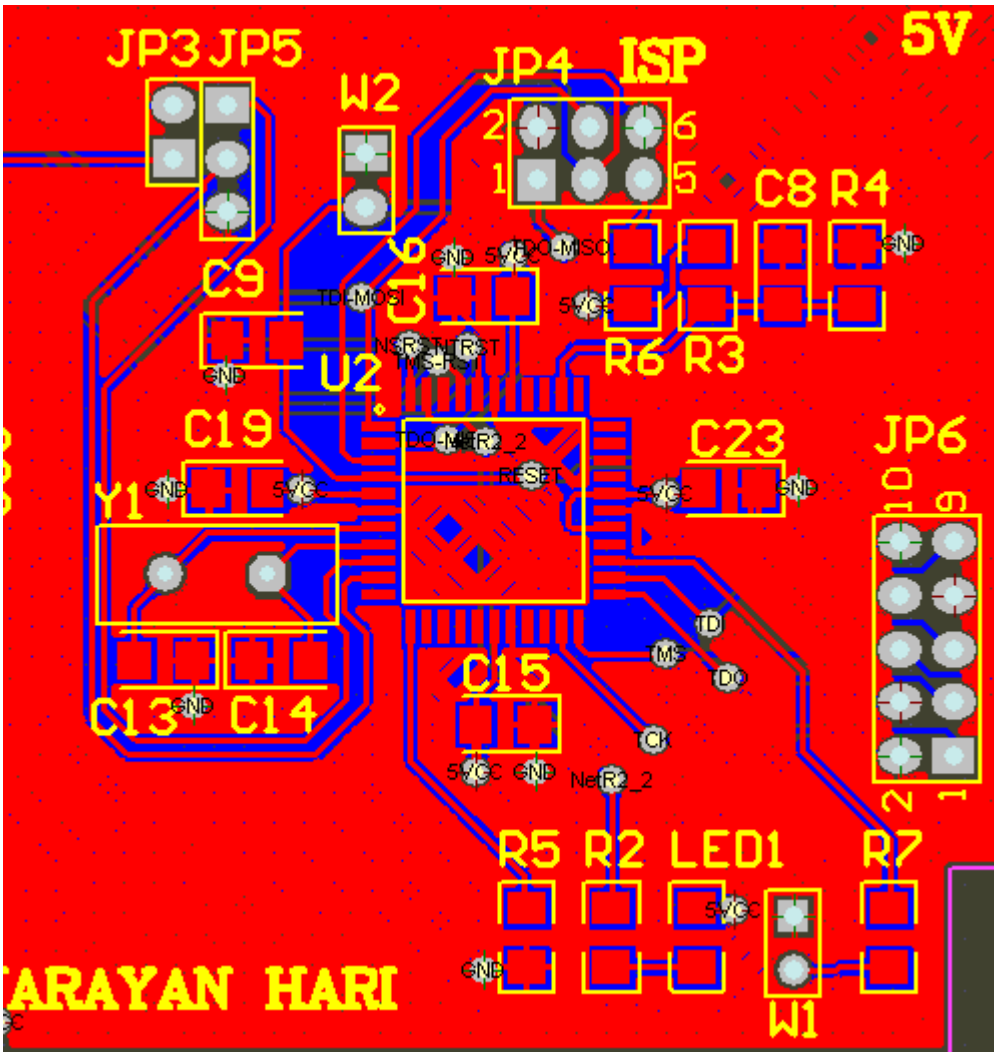


Figure 7.12 ATmega16L PCB routed.

Bootloader firmware development

The bootloader as the name suggest is a program that allows the user for programming the target application program, here JTAG-ICE firmware. The Boot Loader Support provides a real Read-While-Write Self-Programming mechanism for downloading and uploading program code by the MCU itself. This feature allows flexible application software updates controlled by the MCU using a Flash-resident Boot Loader program. The Boot Loader program can use any available data interface and associated protocol to read code and write (program) that code into the Flash memory, or read the code from the Program memory. The program code within the Boot Loader section has the capability to write into the entire Flash, including the Boot Loader memory. The Boot Loader can thus even modify itself, and it can also erase itself from the code if the feature is not needed anymore. The size of the Boot Loader memory is configurable with Fuses and the Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection. The details regarding the Boot Loader support is described in detail from page 246 in the datasheet of ATmega16L.

The bootloader is designed from the reference of bootloader available for AVR Application Note 910 In-System Programming that supports the AVRProg software available in the AVR Studio.

The firmware for bootloader can be understood using the flowchart and the coding is given in the CD-ROM. This bootloader communicates with the AVRProg software.

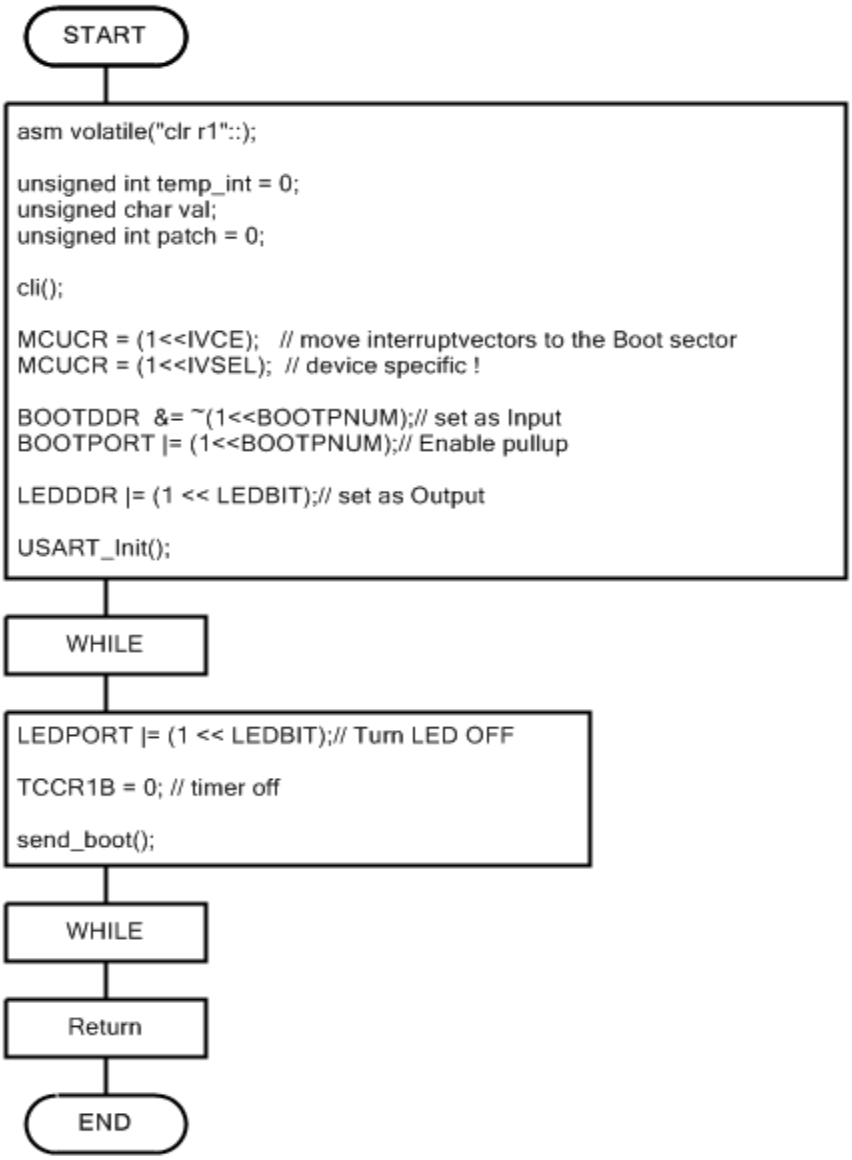


Figure 7.13 Flowchart 1 for Bootloader

The flowchart figure 7.13 shows that there are two while loops, one is for testing the pin PC6 for ISP programming and sending the control to the next while loop for communicating with the AVRProg or to the application section. The UART and the I/O pins are configured along with the timer needed. First of all the control enters the first while loop are flickers the JTAG LED, it stays

in this condition until the PC6 is not pulled low.

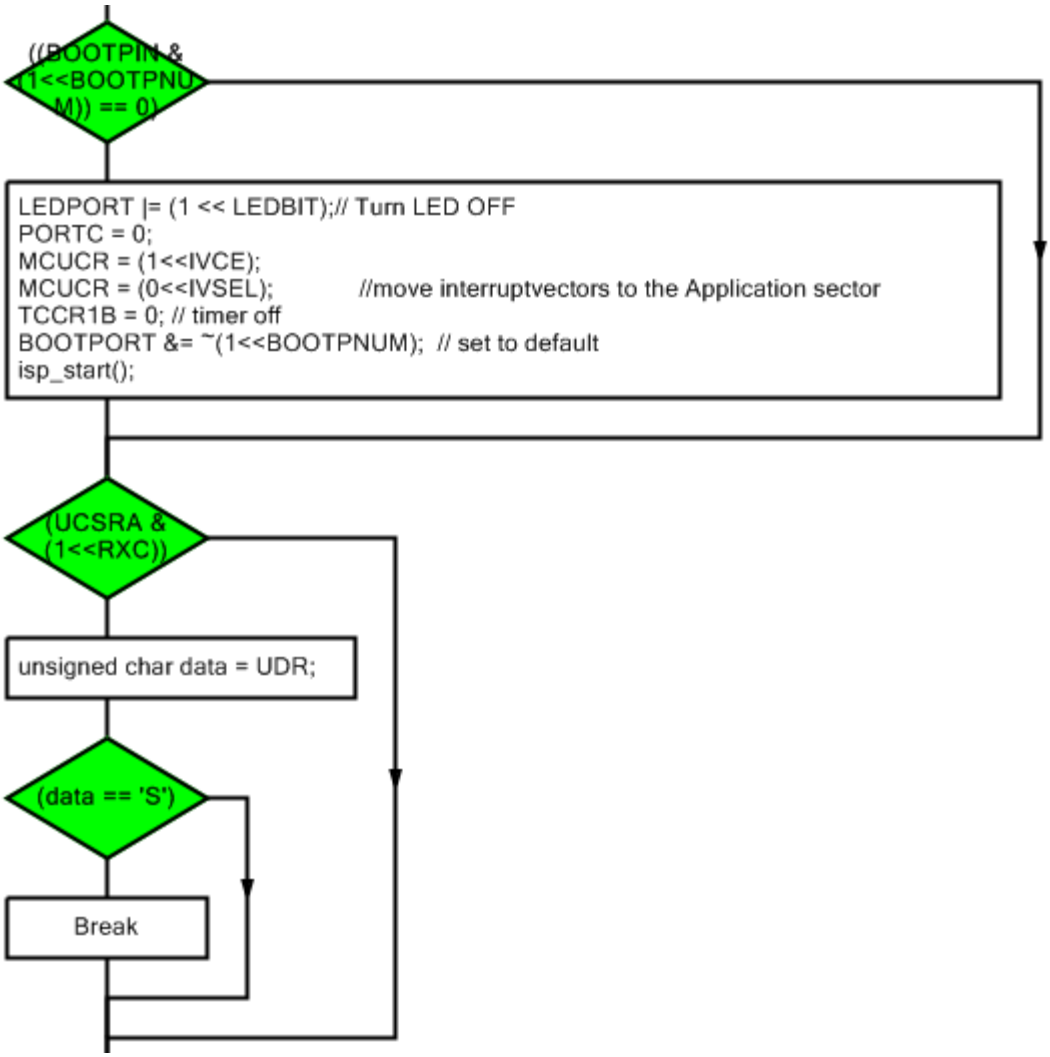


Figure 7.14 Flowchart 2 for Bootloader

The figure 7.14 shows the first half details of first while loop. It can be noted that this loop is scanning status of Pin PC6 and the UART data received. If the PC6 is pulled low and read as logic low then the JTAG LED will go off and interrupts will move to Programming section and trigger the start of ISP. Or if received ‘S’ from the UART control will enter the programming section.

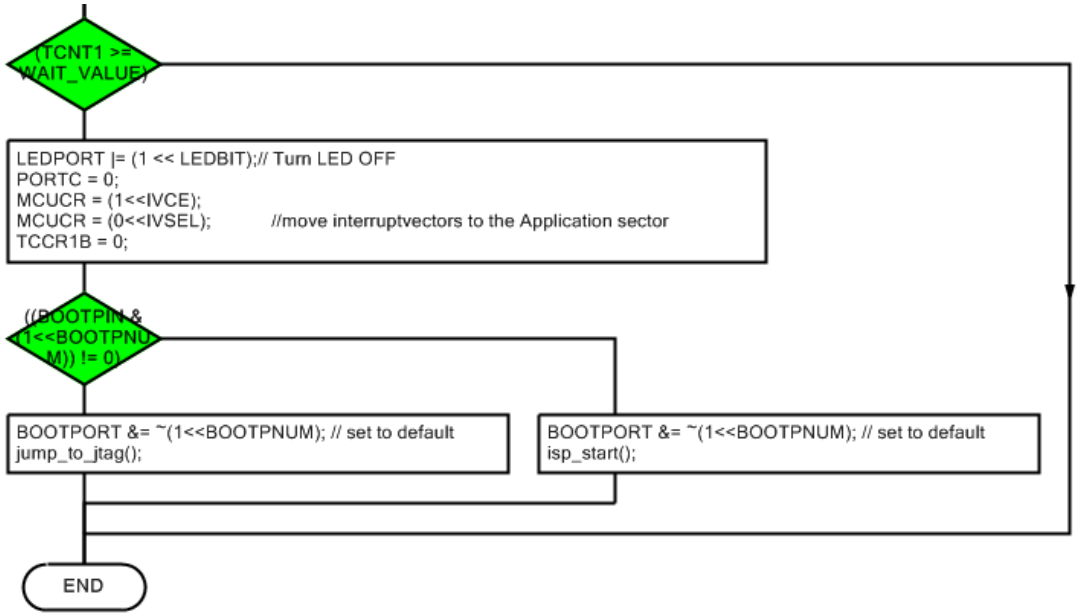


Figure 7.15 Flowchart 3 for Bootloader

The figure 7.15 checks the timer wait time, if it has reached then JTAG LED will turn off and again check for PC6 pin and accordingly go to the application section or programming section. The jump_to_jtag() function passes the control to the JTAG application firmware section, and isp_start() function passes the control to the assembler file Jtag_avr910.S. This file in written in assembly for faster execution of ISP.

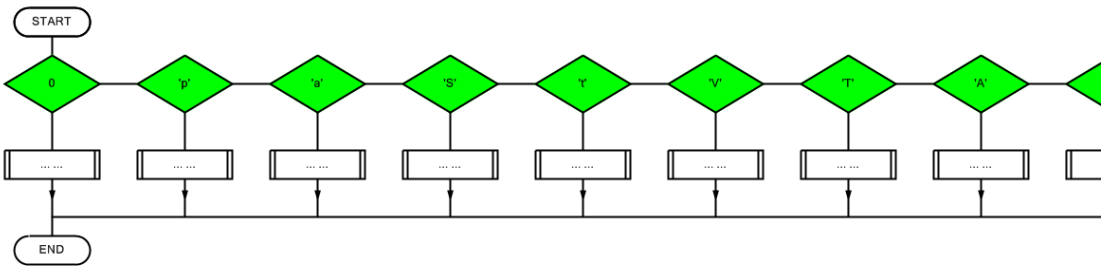


Figure 7.16 Flowchart 4 for Bootloader

The second while loop as in figure 7.16 does not show all the functions but has the following functions as depicted in table. All commands start with a single letter. The programmer returns 13d (carriage return) or the requested data after the command is finished. Unknown commands are replied with a “?”. The reference to these commands is AVR Application Note 109.

Commands	**Host writes**		**Host reads**		
	ID	data	data		Note
Enter programming mode	'P'			13d	1
Report auto increment address	'a'			'Y'	
Set address	'A'	ah al		13d	2
Write program memory, low byte	'c'	dd		13d	3
Write program memory, high byte	'C'	dd		13d	3
Issue Page Write	'm'			13d	
Read program memory	'R'		dd(dd)		4
Write data memory	'D'	dd		13d	
Read data memory	'd'		dd		
Chip erase	'e'			13d	
Write lock bits	'l'	dd		13d	
Write fuse bits	'f'	dd		13d	11
Read fuse and lock bits	'F'		dd		11
Leave programming mode	'L'			13d	5
Select device type	'T'	dd		| 13d |	6
Read signature bytes	's'		3*dd		
Return supported device codes	't'		n*dd	00d	7
Return software identifier	'S'		s[7]		8
Return software version	'V'		dd dd		9
Return hardware version	'v'		dd dd		9
Return programmer type	'p'	|	dd		10
Set LED	'x'	dd		13d	12
Clear LED	'y'	dd		13d	12
Universal command	':'	3*dd	dd	13d	
New universal command	'.'	4*dd	dd	13d	
Special test command	'Z'	2*dd	dd		

Table 7.2 AVRProg Command Table

NOTE 1

The Enter programming mode command MUST be sent one time prior to the other commands, with the exception of the 't', 'S', 'V', 'v' and 'T' commands. The 'T' command must be sent before this command (see note 6).

For programmers supporting both parallel and serial programming mode this command enters parallel programming mode. For programmers supporting only serial programming mode, this command enters serial programming mode.

NOTE 2

The ah and al are the high and low order bytes of the address. For parallel programmers this command issues the Load Address Low/High Byte command. For serial programmers the address byte is stored for use by the Read/Write commands.

NOTE 3

For parallel programmers this command issues the Program Flash command. For serial programmers this command issues the Write Program Memory Command. For devices with byte-wide program memories only the low byte command should be used.

NOTE 4

The contents of the program memory at the address given by the 'A' command are written to the serial port in binary form. For byte wide memories one byte is written. For 16 bit memories two bytes are written, MSB first.

NOTE 5

This command must be executed after the programming is finished.

NOTE 6

The select device type command must be sent before the enter programming command.

NOTE 7

The supported device codes are returned in binary form terminated by 0x00.

NOTE 8

This return a 7 character ASCII string identifying the programmer. For the PCI development board it is "AVR DEV", for the parallel programmer it is "AVR PPR" and for the in-curcuit programmer it is "AVR ICP".

NOTE 9

The software/hardware versions are returned as two ASCII numbers.

NOTE 10

This command should be used to identify the programmer type. The return value is 'S' for serial (or SPI) programmers or 'P' for parallel programmers.

NOTE 11

The write fuse bits command are available only on parallel programmers and only for AVR devices (device code < 0x80). The host should use the return programmer type command to determine the programmer type, do not use the "AVR PPR" idenifier because other programmers may be available in the future.

NOTE 12

Currently only the AVR development board has LEDs. The other boards must implement these commands as NOPs.

NOTE 13

Devices using Page Mode Programming write one page of flash memory before issuing a Page Mode Write Pulse.

The commands are not described as they are directly derived from Application Note 910 and 109 from Atmel.

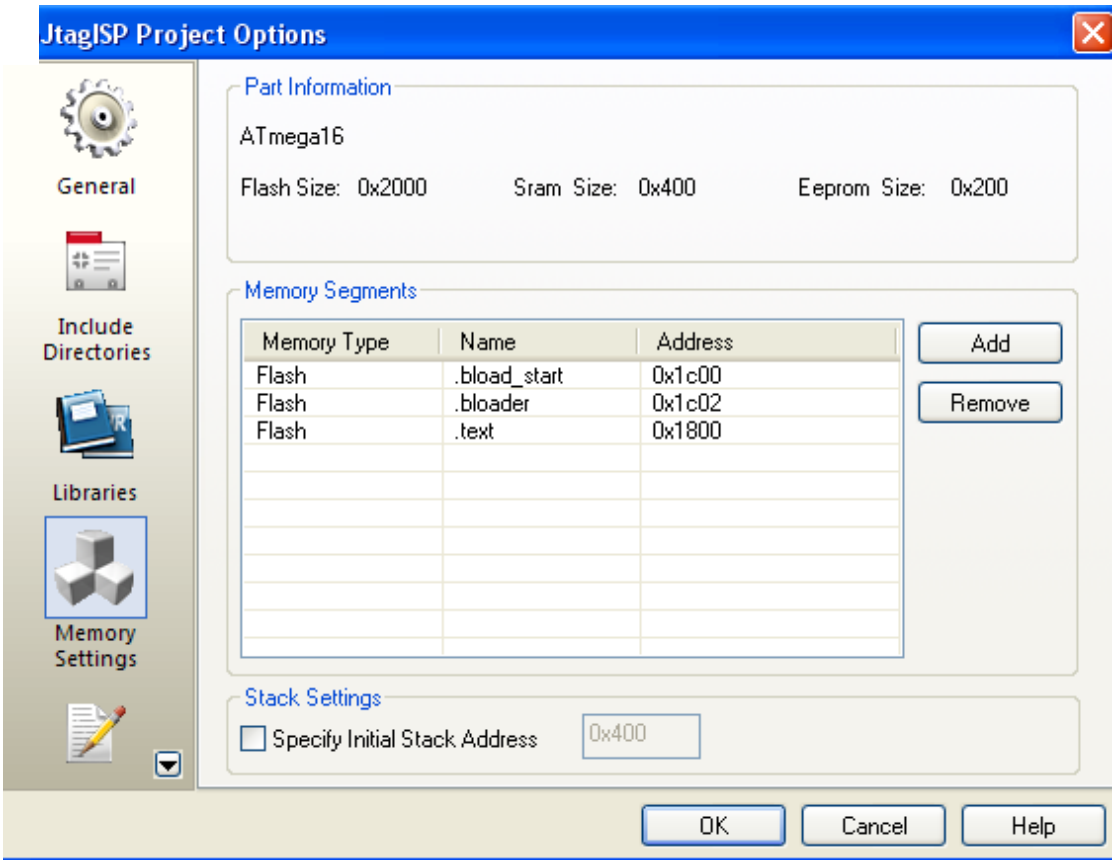


Figure 7.17 Memory section for bootloader Program in AVR Studio.

The figure 7.17 shows the sections of bootloader utilized defined by prototypes in the file bootloader.c and corresponding file Jtag_avr910.S. This

firmware project the good example of C and assembly combinations. After building the project the following message is displayed

Build started 26.2.2011 at 18:44:17

```
avr-gcc    -mmcu=atmega16  -mmcu=atmega16  -Wall  -gdwarf-2    -Os  -
funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums -MD -
MP -MT JTag_avr910.o -MF dep/JTag_avr910.o.d  -x assembler-with-cpp -
Wa,-gdwarf2 -c  ../JTag_avr910.S
avr-gcc    -mmcu=atmega16 -Wall -gdwarf-2    -Os  -funsigned-char  -
funsigned-bitfields   -fpack-struct   -fshort-enums   -MD   -MP   -MT
bootloader.o -MF dep/bootloader.o.d  -c  ../bootloader.c
../bootloader.c: In function 'main':
../bootloader.c:59: warning: unused variable 'patch'
avr-gcc     -mmcu=atmega16     -Wl,-Map=JtagISP.map     -Wl,-section-
start=.bload_start=0x3800   -Wl,-section-start=.bloader=0x3804   -Wl,-
section-start=.text=0x3000  JTag_avr910.o  bootloader.o           -o
JtagISP.elf
avr-objcopy  -O  ihex  -R  .eeprom  -R  .fuse  -R  .lock  -R  .signature
JtagISP.elf JtagISP.hex
avr-objcopy  -j  .eeprom  --set-section-flags=.eeprom="alloc,load"  --
change-section-lma .eeprom=0 --no-change-warnings -O ihex JtagISP.elf
JtagISP.eep || exit 0
avr-objdump -h -S JtagISP.elf > JtagISP.lss

AVR Memory Usage
----------------
Device: atmega16
Program:    1062 bytes (6.5% Full)
(.text + .data + .bootloader)
Data:        206 bytes (20.1% Full)
(.data + .bss + .noinit)
```

Build succeeded with 1 Warnings...

The detail usage of JTAG-ICE is described in section 9.3 of chapter 9.

7.4 Prototype design of ATmega128 (PCB)

Circuit Design

The target microcontroller is AVR ATmega128; it is mounted on the PCB with footprint of QPF 64 pins. This version of ATmega128 works from 0 to 16MHz range so the crystal of 16MHz can be used. Looking at the circuit diagram in

figure 7.18 we find seven different headers are placed for interfacing the ATmega128. The lists of headers with their purpose are given in table. The headers listed here give the power to use ATmega128 with its full features.

Header name	**Connectivity**	**Purpose**
JP7	ATmega16L	For JTAG programming / debugging
JP8	Ext2. With Power supply	PORT B,E,F extended with +12V, -12V,+5V, +3.3V DC
JP9	UART1	Used for external INT2,INT3 or UART1 port
JP10	RAM	Used for selection of external SRAM
JP11	PORT E,B	PORTE and some of PORTB with power supply
JP12	PORTD,B	PORTD and some of PORTB with GND and reset
JP13	Ext1	Combination of JP11 and JP12

Table 7.3 Header of ATmega128 Prototype.

A 32.768 KHz clock is connected to pin 18 PG3 and pin 19 PG4 for internal real time clock and clock synchronization. The chip has large number of I/O pins i.e. 53 pins for interfacing parallel and serial memory, ADC, PWN, external Interrupts, UARTs. Out of the two UARTs one UART0 is connected with MCS98335CV UARTB and another is connected to JP9 for external use. The interface to the ATmega128 will be discussed in section 7.5 of this chapter.

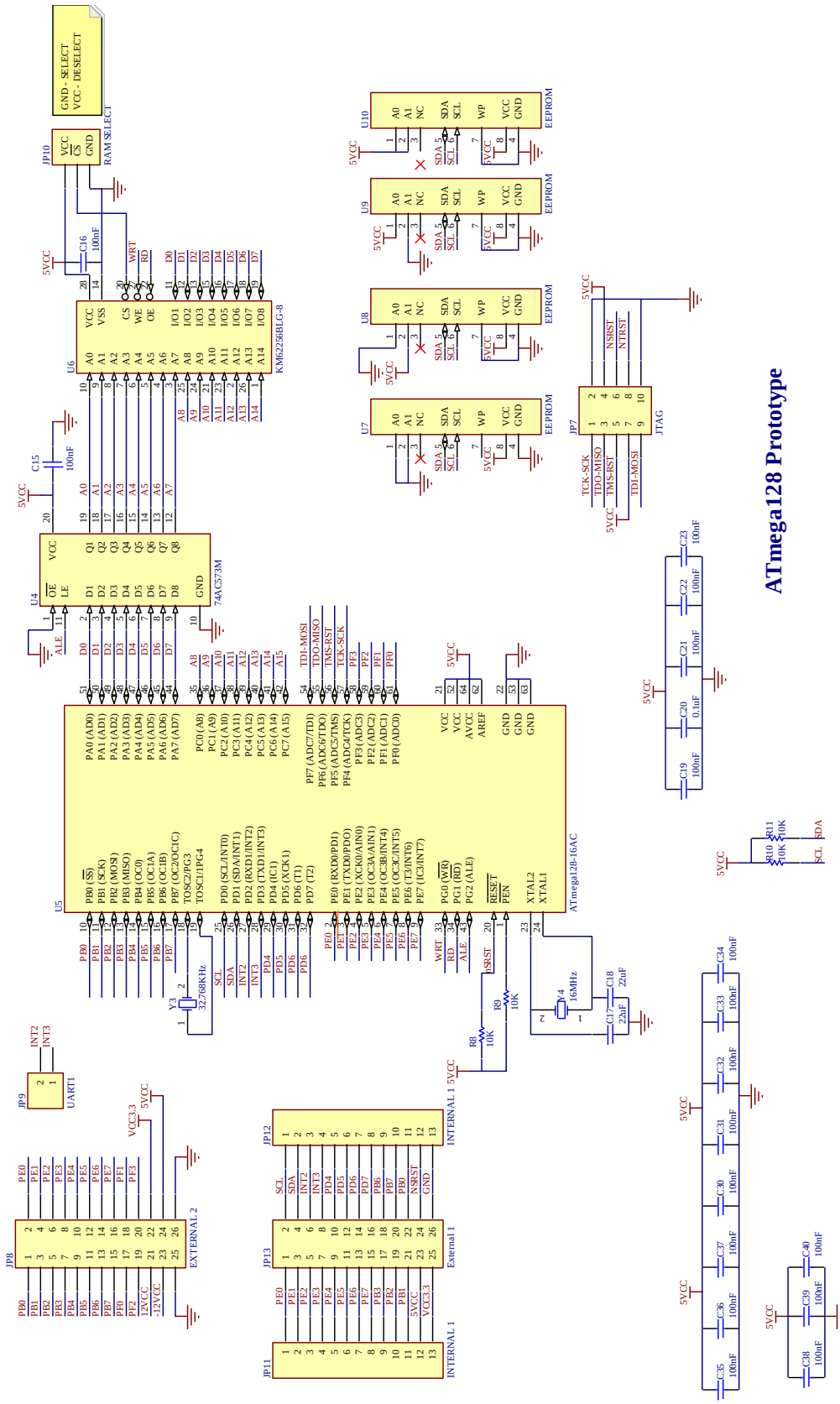


Figure 7.18 Prototype of ATmega128 and interface.

PCB Design

As ATmega128 is the target microcontroller with large I/O pins. The interfacing parts with it require large PCB area. The figure shows part of the PCB routed diagram of ATmega128 and its interfacing peripherals.

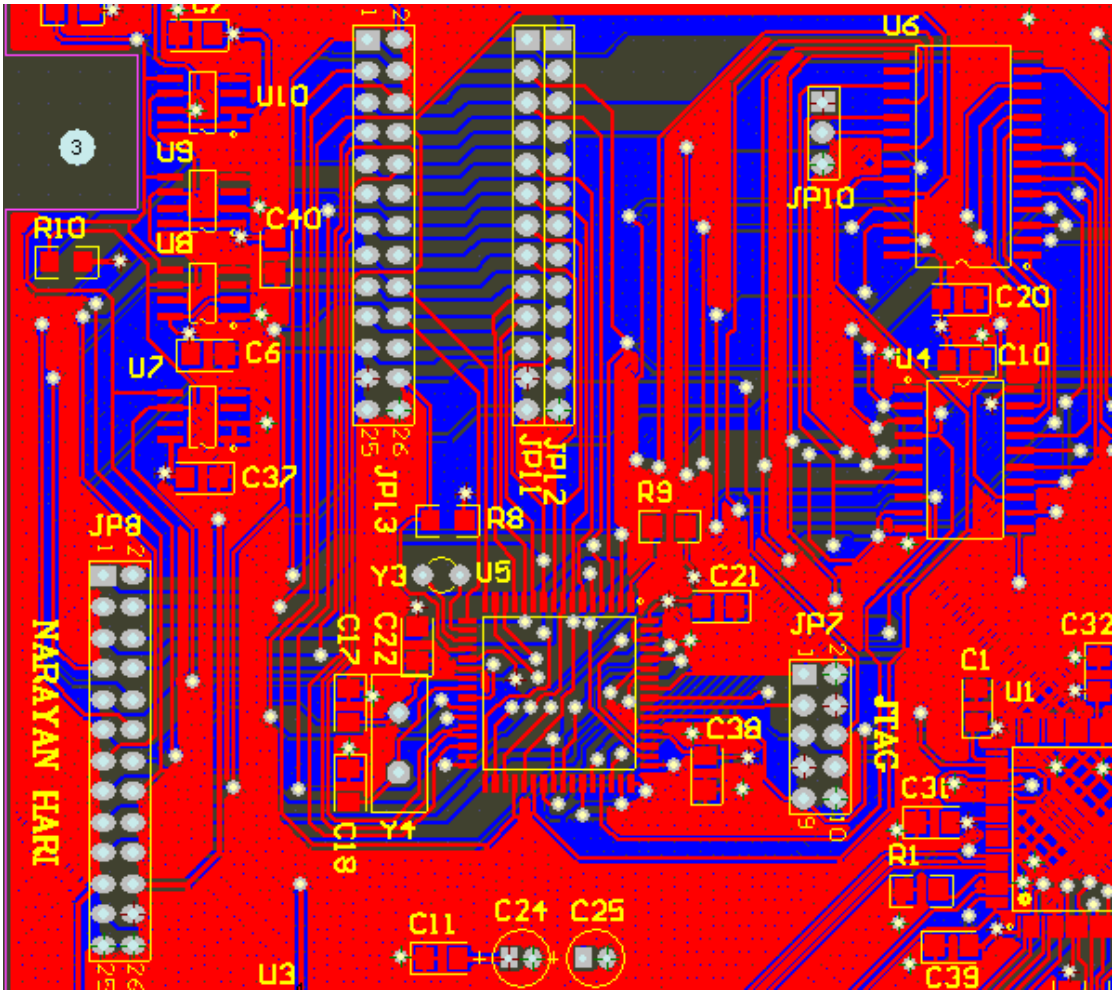


Figure 7.19 ATmega128 PCB routed with peripherals

Program code for testing the circuit figure is given in appendix sample exercise. The exercise is given for ATmega16L but can be easily ported to ATmega128 by just changing the required Port pins. Programs in C and assembly are developed and tested on this PCI card successfully. The PCI card PCB has also a 3.3V regulated out voltage on side of prototype area as shown in figure 7.29.

7.5 Interfacing Modules (PCB)

Interfacing modules gives the capability of testing the internal peripheral functioning on the ATmega128. All in all there are 6 modules developed and tested with firmware.

1. Basic Input/Output Module with external interrupts.
2. PWM control interface via UART0 Module.
3. I^2C interface EEPROM Module.
4. Parallel interface SRAM Module.
5. ADC Module via UART0.
6. 1-wire iButton interface.

The functioning of ATmega128 is no limited to the above modules, but external circuit can be built up on the free prototyping space along with power supply.

7.5.1 Basic I/O Module with external interrupts.

Ports in the AVR are gates from the central processing unit to internal and external hard- and software components. The CPU communicates with these components, reads from them or writes to them, e.g. to the timers or the parallel ports. The most used port is the flag register, where results of previous operations are written to and branch conditions are read from.

There are 64 different ports, which are not physically available in all different AVR types. Depending on the storage space and other internal hardware the different ports are either available or accessible or not. Which of these ports can be used is listed in the data sheets for the processor type. Ports have a fixed address, over which the CPU communicates. The address is independent from the type of AVR. So e.g. the port address of port B is always 0x18 (0x stands for hexadecimal notation). It is not required to remember these port addresses, they have convenient aliases. These names are defined in the include files (header files) for the different AVR types, that are provided from the producer. The include files have a line defining port B's address as follows:

.EQU PORTB, 0x18

So just remember the name of port B, not its location in the I/O space of the chip. The include file m128def.inc is involved by the assembler directive

.INCLUDE " C:\Program Files\Atmel\AVR Tools\AvrAssembler2\Appnotes \8515def.inc". the standing instruction to setup the project for assembly and C will be explained in section 9.2 of chapter 9.

Circuit for I/O with external Interrupts is shown in the figure 7.20. The circuit is connected to the header Ext2 JP8, where the output port will be PORTB and input port will PORTE. Apart from the I/O functions the PORTB also has alternate functions when this port is not used as general I/O port is listed in table 30 of page 74 of ATmega128 datasheet. Four PWM or timer/counter modes and SPI interface can be used at header JP8. Apart from the I/O functions the PORTE also has alternate functions when this port is not used as general I/O port is listed in table 39 of page 81 of ATmega128 datasheet. The pins of PORTE can be used as PWM or timer/counter, external interrupt, Analog comparator and UART0 peripherals.

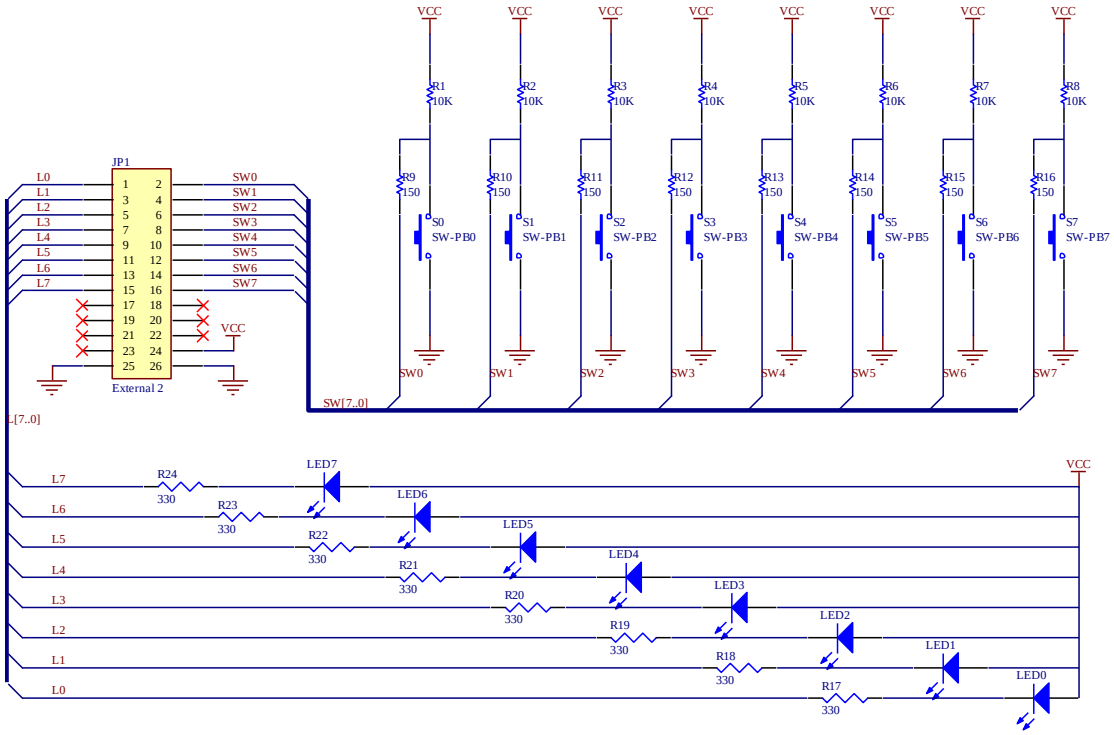


Figure 7.20 PCI external 2 card circuit schematic.

The flowchart figure 7.21 shows the LEDs shifting towards right after some delay at PORTB of ATmega128. The DDRB is first written with 0xFF to

configure it as output port, so that anything loaded to PORTB is reflected on the LEDs. This program prepared using Flowcode V4 for AVR from Matrix Multimedia.

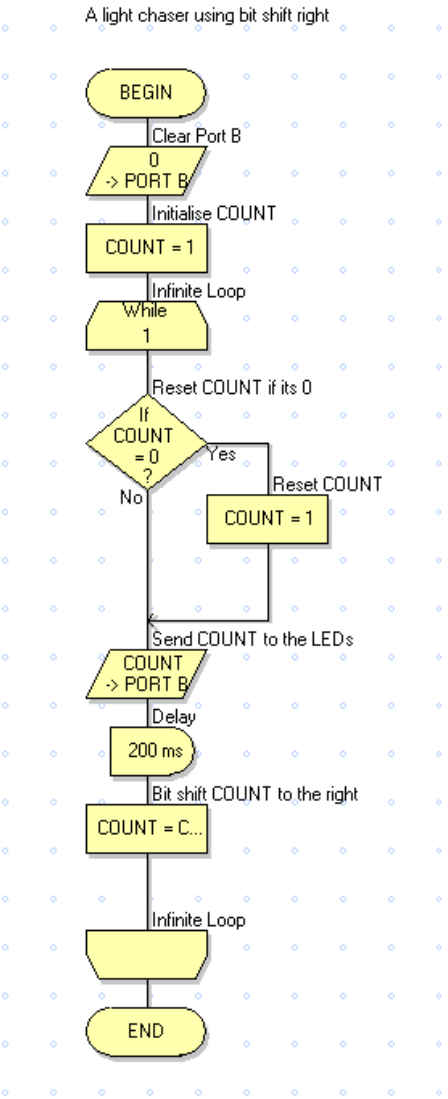


Figure 7.21 flowchart of I/O module.

An example with interrupts is taken from paper **Atmel AVR ATtiny15L Interrupt Concepts, Programming and Utilization** published in **Lab Experiments Journa**l, December – 2007 Issue. The assembly program given below checks the logic low at pin E0 and sends the status to output at PORTB0.

```
;THIS PROGRAM CHECKS THE LOW LOGIC OF PINE0.
.INCLUDE "M128DEF.INC"
.CSEG
```

```
.ORG 0X00
          RJMP TODO
.ORG 0X050
TODO:     CBI DDRE,0      ;SET DDRE0 TO 0
          SBI PORTE,0     ;SET PORTE0 TO 1
          SBI DDRB,0      ;SET DDRB0 TO 1
TT:       SBI PORTB,0     ;SET PORTB0 TO 1
AAA:      IN R21,PINE     ;CHECK THE PINE0 FOR GROUND
          NOP
          BST R21,0
          BRTS TT         ;IF PINE0 IS 1 THEN SET PB1 TO 1
          CBI PORTB,PORTB0;ELSE CLEAR PBO TO 0
          RJMP AAA
.EXIT
```

;ASSEMBLE USING AVR STUDIO 4

The program listing shown here always polling the input pin PE0 for logic levels, but if waiting for the pin is not feasible as the CPU is executing some other tasks, then the interrupts come in picture.

The flowchart for the interrupt sense program is figure 7.22. After the initializations of interrupts and sleep modes, the CPU enters the sleep mode, if the pin PE4 is pressed, interrupt sense low is generated and the controls executes the ISR that blinks all the LEDs of PORTB.s

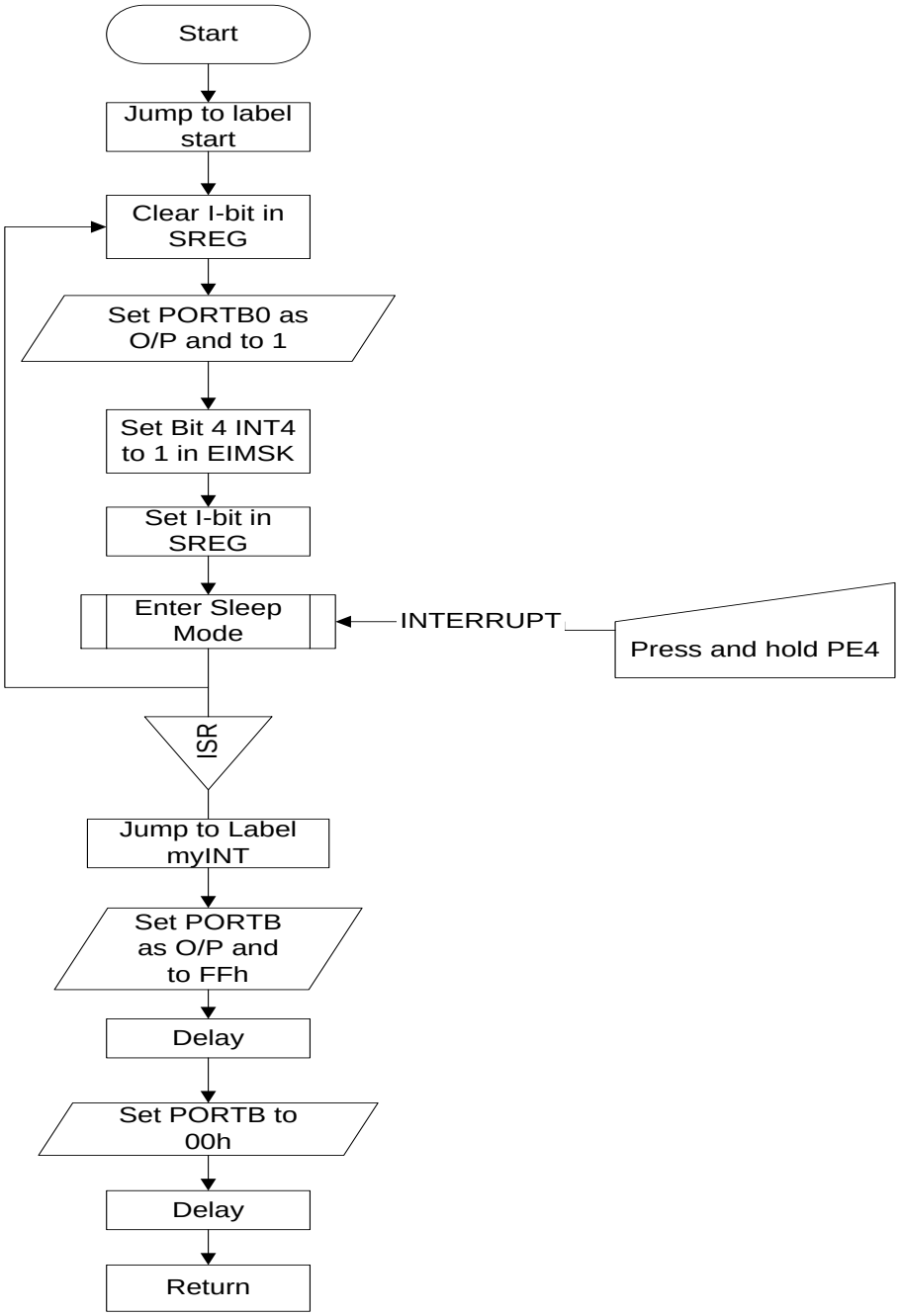


Figure 7.22 flowchart for I/O external interuupts

7.5.2 PWM control interface via UART0 Module

PWM combined with an analog filter can be used to generate analog output signals, i.e. a digital to analog converter (DAC). A digital pulse train with a constant period (fixed base frequency) is used as a basis. To generate different analog levels, the duty cycle and thereby the pulse width of the digital signal is changed. If a high analog level is needed, the pulse width is increased and vice versa.

Averaging the digital signal over one period (using an analog low-pass filter) generates the analog signal. A duty cycle of 50% gives an analog signal with half the supply voltage, while 75% duty cycle gives an analog signal with 75% supply voltage. The Pin PB4 acts as OC0 when using it as PWM output. The DDRB4 must be set 1 for output. The required PWM waveform is PWM, phase correct.

The phase correct PWM mode (WGM01:0 = 1) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is based on a dual-slope operation. The counter counts repeatedly from BOTTOM to MAX and then from MAX to BOTTOM. In noninverting Compare Output mode, the output compare (OC0) is cleared on the compare match between TCNT0 and OCR0 while counting up, and set on the compare match while down counting.

In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications. The PWM resolution for the phase correct PWM mode is fixed to 8 bits. In phase correct PWM mode the counter is incremented until the counter value matches Max. When the counter reaches MAX, it changes the count direction. The TCNT0 value will be equal to MAX for one timer clock cycle.

The assembly program for demonstration of PWM connecting to LED for dimming is given in the sample exercise G.5. The I/O interface can be used for demonstration as PB4 has LED4 connected to it. The UART0 receives the data from the PC serial port to change the duty cycle of the pulse.

7.5.3 I^2C interface EEPROM Module.

To demonstrate the use of I^2C peripheral in ATmega128, four AT24C512 I^2C EEPROMs are cascaded to have continuous large storage area. The figure 7.18 shows that U7, U8, U9 and U10 chips are connected to the I^2C bus of ATmega128. The AT24C512 provides 524,288 bits of serial electrically erasable and programmable read only memory (EEPROM) organized as 65,536 words of 8 bits each. The device's cascadable feature allows up to four devices to share a common two-wire bus. The 512K is internally organized as 512 pages of 128-bytes each. Random word addressing requires a 16-bit data word address. The setup of EEPROM memory on the PCI card is given in table 7.4. The result is 2Mbytes of data on PCI card.

AT24C512	A1	A0	Address Range(H)	Bytes
U7	0	0	0 - FFFF	64K
U8	0	1	10000 - 1FFFF	64K
U9	1	0	20000 – 2FFFF	64K
U10	1	1	30000 – 3FFFF	64K

Table 7.4 EEPROMs on PCI card

The write protect input, when connected to GND, allows normal write operations. When WP is connected high to VCC, all write operations to the memory are inhibited. If the pin is left floating, the WP pin will be internally pulled down to GND if the capacitive coupling to the circuit board VCC plane is <3 pF. If coupling is >3 pF, Atmel recommends connecting the pin to GND. Switching WP to VCC prior to a write operation creates a software write protect function. The sample program G.8 for reading and writing some data on specific addresses from UART0 of ATmega128 using PC serial port is given in the sample exercises using BASCOM-AVR complier.

7.5.4 Parallel interface SRAM Module.

The parallel Address/data multiplexed bus with ALE, /RD, /WR similar buses of 8085/86 are also implemented with ATmega128. The External Memory Interface provides, it is well suited to operate as an interface to memory devices such as External SRAM and Flash, and peripherals such as LCD display, A/D, and D/A. The main features are:

- Four different wait-state settings (including no wait-state).
- Independent wait-state setting for different extErnal Memory sectors (configurable sector size).
- The number of bits dedicated to address high byte is selectable.
- Bus-keepers on data lines to minimize current consumption (optional).

When the eXternal MEMory (XMEM) is enabled, address space outside the internal SRAM becomes available using the dedicated External Memory pins. The interface consists of:

• AD7:0: Multiplexed low-order address bus and data bus.

• A15:8: High-order address bus (configurable number of bits).

• ALE: Address latch enable.

• RD: Read strobe.

• WR: Write strobe.

The XMEM interface will auto-detect whether an access is internal or external. If the access is external, the XMEM interface will output address, data, and the control signals on the ports according to Figure 13 (this figure shows the wave forms without wait-states). When ALE goes from high-to-low, there is a valid address on AD7:0. ALE is low during a data transfer. When the XMEM interface is enabled, also an internal access will cause activity on address, data and ALE ports, but the /RD and /WR strobes will not toggle during internal access. When the External Memory Interface is disabled, the normal pin and data direction settings are used. Note that when the XMEM interface is disabled, the address space above the internal SRAM boundary is not mapped into the internal SRAM. Figure 12 illustrates how to connect an external SRAM to the AVR using an octal latch (typically “74 x 573” or equivalent) which is transparent when G(/OE) is high.

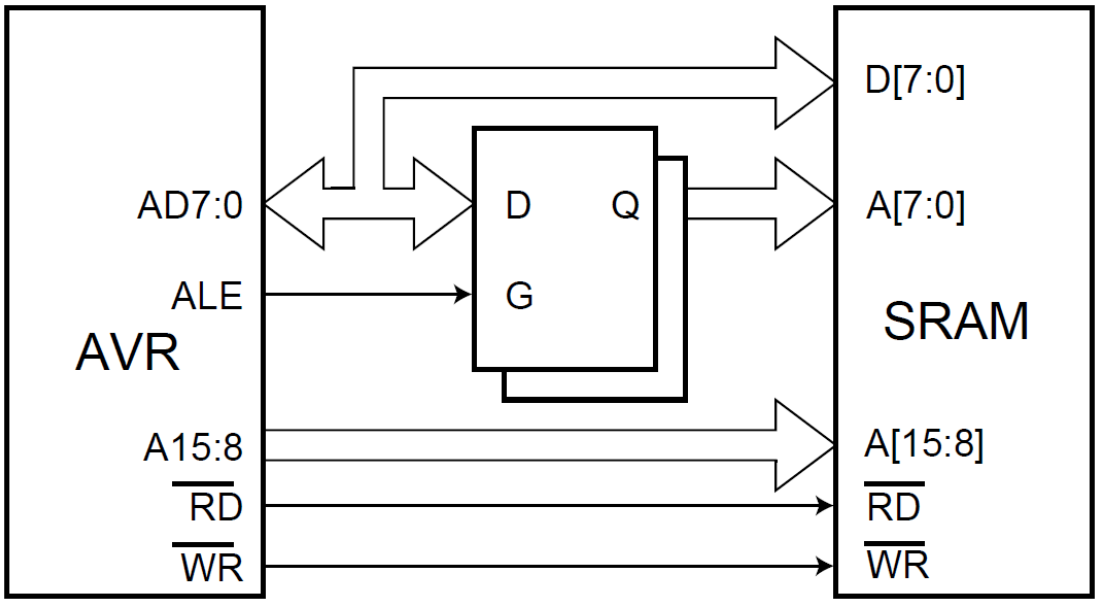


Figure 7.23 Example of External SRAM connected to AVR

The SRE bit of MCUCR is to be enabled for using XMEM interface. Also the external spaces are divided into several sectors that are described in XMCRA register on page 32 of ATmega128 datasheet. The unused pins of PORTC can used as normal I/O pins if not used by setting the XMCRB register defined on page 33 of ATmega128 datasheet.

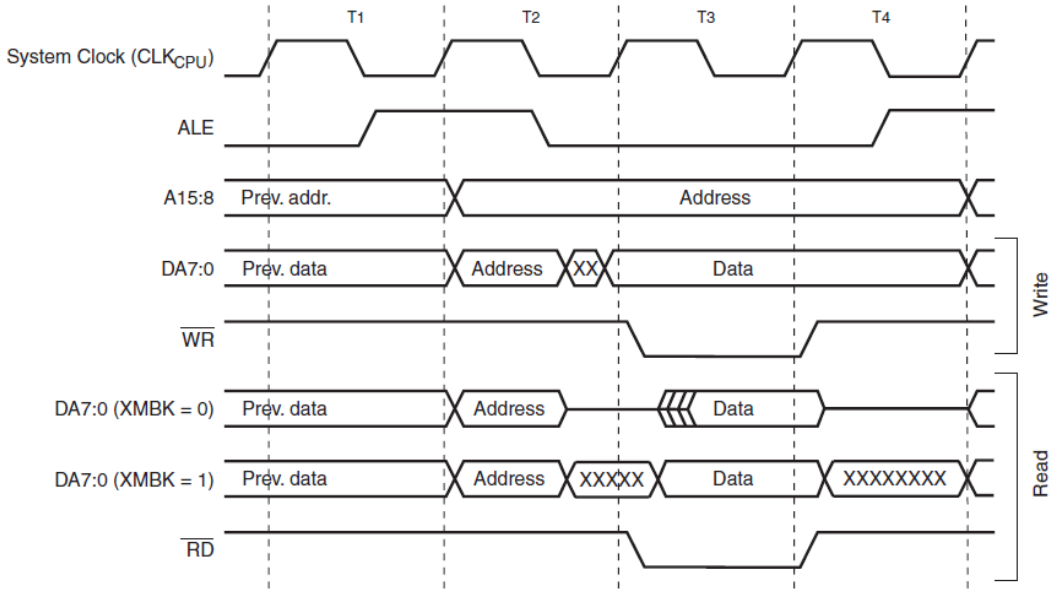


Figure 7.24 Typical Timing Diagram for XMEM

Since the external memory is mapped after the internal memory as shown in Figure 7.25, the external memory is not addressed when addressing the first

4,352 bytes of data space. It may appear that the first 4,352 bytes of the external memory are inaccessible (external memory addresses 0x0000 to 0x10FF). However, when connecting an external memory smaller than 64 KB, for example 32 KB, these locations are easily accessed simply by addressing from address 0x8000 to 0x90FF. Since the External Memory Address bit A15 is not connected to the external memory, addresses 0x8000 to 0x90FF will appear as addresses 0x0000 to 0x10FF for the external memory. Addressing above address 0x90FF is not recommended, since this will address an external memory location that is already accessed by another (lower) address. To the Application software, the external 32 KB memory will appear as one linear 32 KB address space from 0x1100 to 0x90FF.

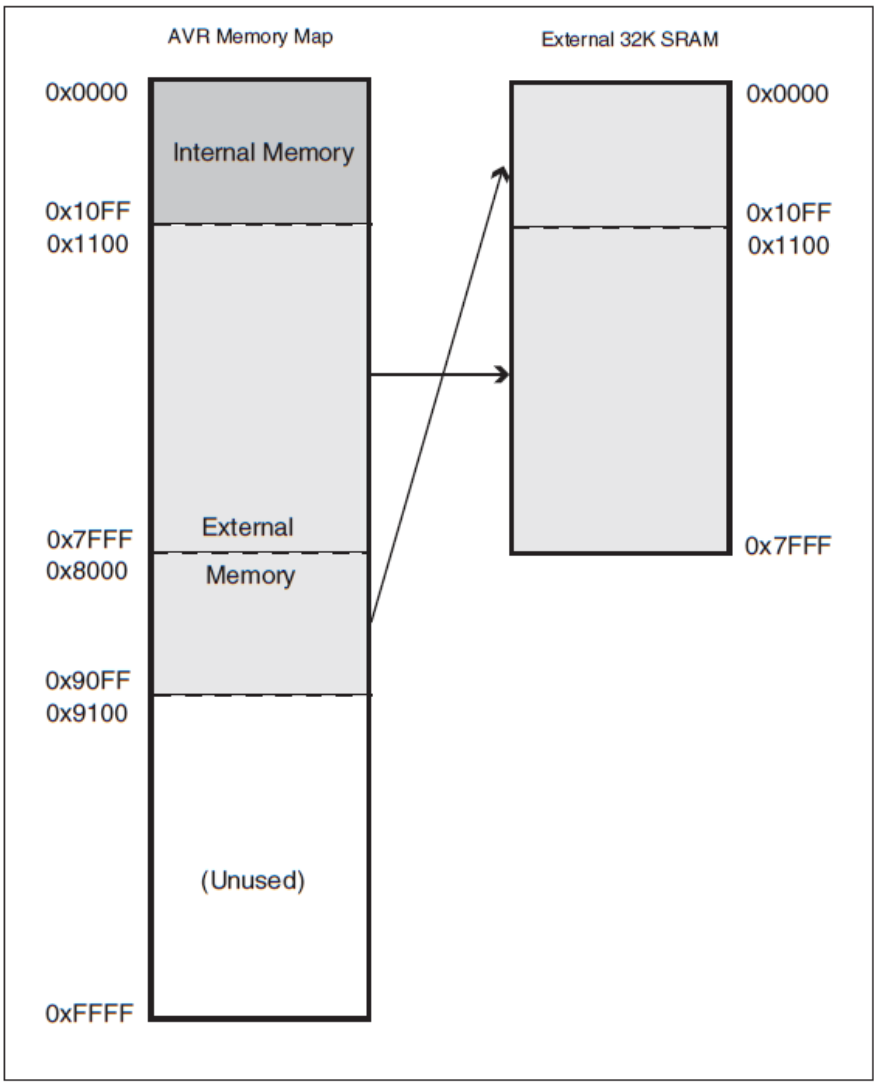


Figure 7.25 External memory Address Usage Diagram.

The following coding in assembly language can be used for accessing the external memory. The circuit diagram for the same is shown in figure.7.18.

```
;This example shows the access to the external memory
;along with internal memory, Here PC7 is released
.include "m128def.inc"

.CSEG
RJMP RESET
.ORG 0X46

RESET:      LDI R16,LOW(RAMEND)           ;SET THE STACK POINTER
            LDI R17,HIGH(RAMEND)
            OUT SPL,R16
            OUT SPH,R17
            LDI  R16,(1<<SRE); enable the Xmen interface
            OUT MCUCR,R16
            LDI R16,(1<<XMM0); PC7 is released due to 32K SRAM
            STS XMCRB, R16
            LDI R16, $A3
            STS $204, R16;accessing memory 0x204 of internal SRAM
            LDS  R18,$204
            com r16
            STS $8204, R16;accessing memory 0x204 of external SRAM
            LDS  R18,$8204
            inc r16
            STS $7204, R16;accessing memory 0x7204 of external SRAM
            LDS  R18,$7204
            RJMP RESET
.EXIT
```

7.5.5 ADC Module via UART0.

The ATmega128 has 8 Multiplexed Single Ended Input Channels, 7 Differential Input Channels, 2 Differential Input Channels with Optional Gain of 10x and 200x with resolution of 10-bit and conversion time of 13 - 260µs. The PORTF on external 2 headers JP8 from PF0 to PF5 can be used for interfacing analog circuit. The voltage range is from 0 to AVcc and different selectable reference voltages as mentioned in the datasheet.

The ADC in Atmel's AVR devices can be configured for single-ended or differential conversion. Single-ended mode is used to measure input voltages

on a single input channel, while differential mode is used to measure the difference between two channels. Regardless of conversion mode, input voltages on any channel must stay between GND and AVcc.

When using single-ended mode, the voltage relative to GND is converted to a digital value. Using differential channels, the output from a differential amplifier (with an optional gain stage) is converted to a digital value (possibly negative). To decide the conversion range, the conversion circuitry needs a voltage reference (VREF) to indicate the voltage level corresponding to the maximum output value.

According to the datasheets, VREF should be at least 2.0 V for standard devices, while devices operating down to 1.8V can use a reference voltage down to 1.0V. This applies both to single-ended and differential mode. The 10-bit ADC of the AVR therefore converts continuous input voltages from GND to VREF to discrete output values from 0 to 1023.

Any applied input voltage greater than the reference voltage VREF will return the maximum value (1023 using 10-bit ADC), and any negative input voltage will return 0. When using single-ended mode, the ADC bandwidth is limited by the ADC clock speed. Since one conversion takes 13 ADC clock cycles, a maximum ADC clock of 1MHz means approximately 77k samples per second. This limits the bandwidth in single-ended mode to 38.5 kHz, according to the Nyquist sampling theorem.

When using differential mode, the bandwidth is limited to 4 kHz by the differential amplifier. Input frequency components above 4 kHz should be removed by an external analog filter, to avoid non-linearity. The input impedance to VCC and GND is typically 100 MΩ. Together with the output impedance of the signal source, this creates a voltage divider. The signal source should therefore have sufficiently low output impedance to get correct conversion results.

Here in the circuit AVcc and AVref are connected to 5Vcc. An example of single ended conversion of voltage range 0 to 5V can be done by connecting one potentiometer at pin 19 PF0 of JP8. The measured voltage is send via UART0 and checked on the PC serial port interface. The sample exercise is included in the appendix. The circuit 7.26 shows how the potentiometer is connected to the ADC pin PF0.

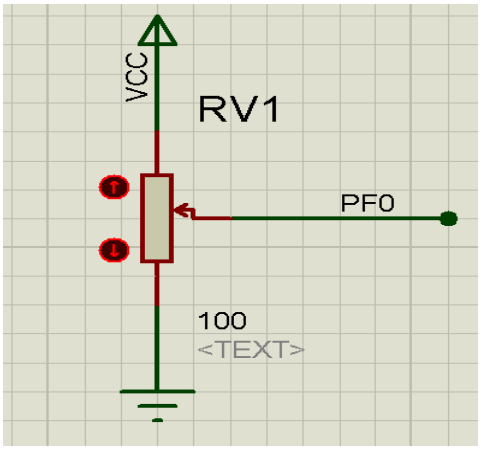


Figure 7.26 Pot Interfacing

7.5.6 1-wire iButton interface.

This module is implemented based on the paper entitled **Secure Digital Access system using iButton** in Construction section of **Electronics for You** magazine, November, 2010 issue.

An iButton[24] device is miniature computer chip enclosed in a 16mm stainless steel case. It communicates with the AVR microcontroller using 1-wire [25] protocol. The iButton devices do not need power supply, rather they use parasite power of the 1-wire through which they interface to AVR with an additional ground line. In 1-wire bus system there is single master and several slaves attached. Here AVR microcontroller using only a single I/O pin works as 1-wire master. All iButton devices have globally unique ROM code laser-printed at manufacturing process with family code and CRC code of them. The figure 7.27 shows sample iButton. The 1-wire protocol strictly uses master-slave scheme and its communication is half-duplex asynchronous.[4].

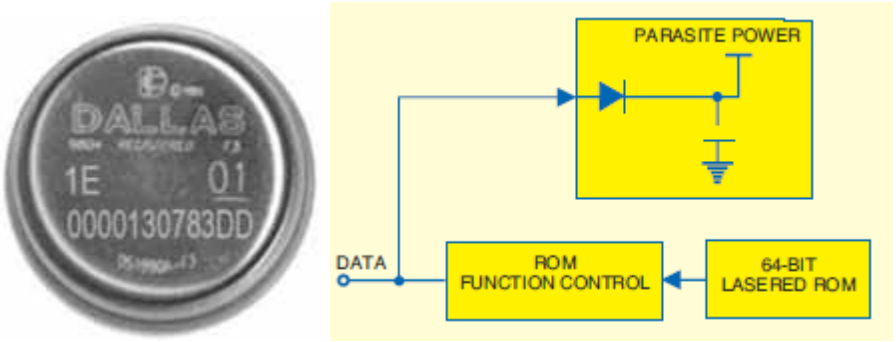


Figure 7.27 iButton and its Block diagram

The figure 7.27 shows the block diagram. The figure 7.28 shows 64-bit Laser ROM of iButton. The 1-wire interface is connected to pin 25 PD0 works as INT0 interrupt. The example program for the iButton is written in BASXOM-AVR and listed in sample example section in appendix.

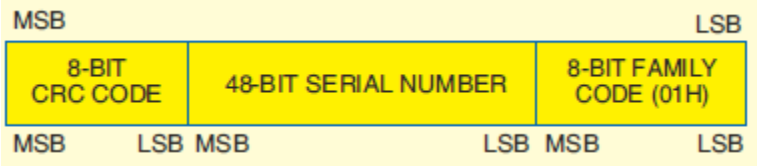


Figure 7.28 64-bit laser ROM of iButton.

Three different 1-Wire implementations are possible: software only (polled), polled UART and interrupt-driven UART. A short description of each is given below.

It is possible to implement the 1-Wire protocol in software only without using any special hardware. This solution has the advantage that the only hardware it occupiesis one general purpose I/O pin (GPIO). Since all GPIO pins on the AVR are bidirectional, and have selectable internal pull-up resistors, the AVR can control a 1- Wire bus with no external support-circuitry. In case the internal pull-up resistor is not suitable with the current configuration of slave devices, only one external resistor is needed. On the downside this implementation relies on busy waiting during “Reset/Presence” and bit signaling. To ensure correct timing on the 1-Wire bus, interrupts must be disabled during transmission of bits. The allowed delay between transmission of two bits (recovery time) has no upper limit, however, so it is safe to handle interrupts after every bit transmission. This makes the worst-case interrupt latency due to 1-Wire bus activity equal to execution time of the “Reset/Presence” signal, less than 1 ms.

The polled UART driver uses the UART module found on many AVRs to generate the necessary waveforms at the bit-level. The rest of the driver is equal to the software only driver. The main advantage with this driver compared to the software only driver is code size and the fact that interrupts do not need to be turned off during bit signaling since the UART module handles the timing details independently. On the downside it requires two GPIO pins and some external support circuitry.

The Interrupt-driven UART driver uses the UART to generate the waveforms in the same way as the Polled UART driver. In addition it takes advantage of the interrupt capabilities found in the UART module to automate sending or receiving of up to 255 bits of data. The application note #214 **Using a UART to Implement 1 – Wire Bus** Master from Maxim is also useful to design such a solution.

The figure 7.29 shows the full routed 4 – layered PCB of the PCI card.

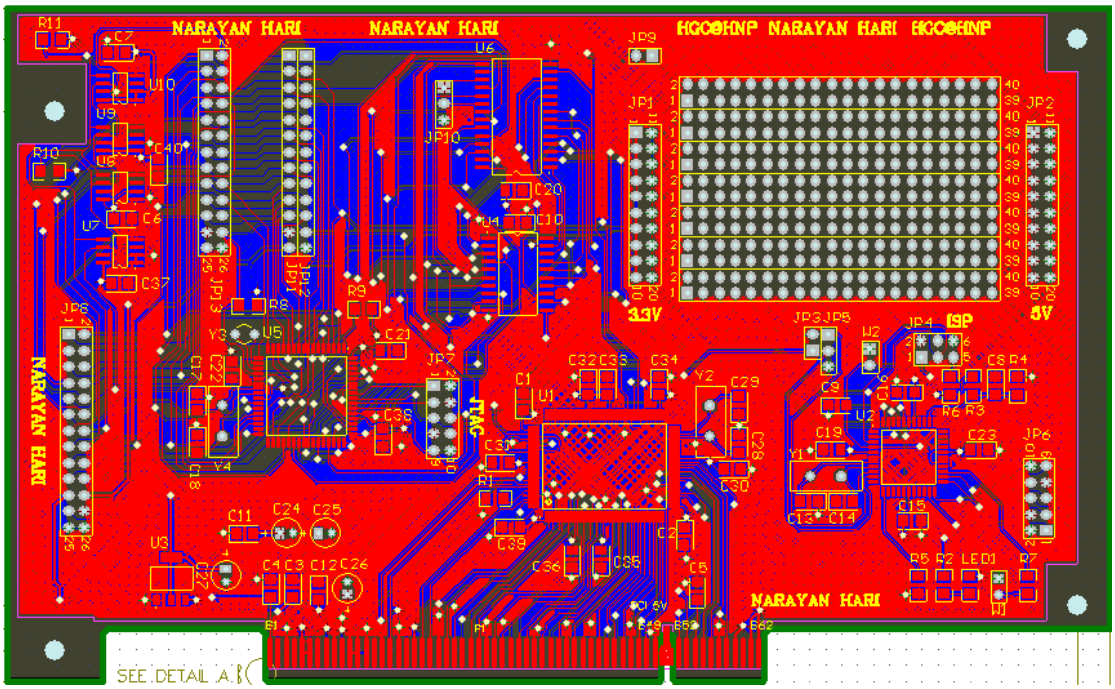


Figure 7.29 Fully routed 4 – layered PCB of PCI card.

Chapter 8

Implementation, Testing and Deployment

Chapter 8 Implementation Testing and Deployment.

The testing strategy for the project is to test all possible test points and to verify functional correctness, and also to test system components iteratively when they are implemented. This chapter combines the implementation testing phase as shown in figure 2.1 of chapter 2. Designing of circuit along with PCB is described in detail design phase and the sample examples of firmware programs are also listed. Having hands on the PCI card repeatedly for proper functioning of this design is illustrated in this chapter. Testing each subsystem and getting satisfactory output result in their integration for full proof working of PCI card based microcontroller based trainer system. Each of the subsystems described in chapter 8 were fabricated as an individual prototype and then the same design was transferred to the PCI card schematic and printed circuit board. The PC with Microsoft Windows XP Professional with SP2/SP3 is used to test the PCI card.

8.1 Testing of PCI Controller subsystem.

The drivers for all the popular OS are available from the website of MosChip Semiconductor Technology. Download the appropriate driver for the working OS. Install the driver as shown in the operating manual of chapter 9. Initially only the MCS9835CV chip is to be tested for working, after then the next subsystem will be tested.

The following steps involved to test the PCI controller subsystem must be followed:

Install the device drivers for MCS9835CV and reboot the PC, do check the device manager configuration as shown in figure 8.1.

1. Solder the MCS9835CV chip in the area provided along with the 22.1184 MHz crystal and the resistors and decoupling capacitors as shown in figure 7.9 of chapter 7. The figure 8.2 shows the only the PCI chip soldered. Here it important to solder the header JP3 which is used for testing purpose here, so that it can be shorted by inserting 2-pin jumper.

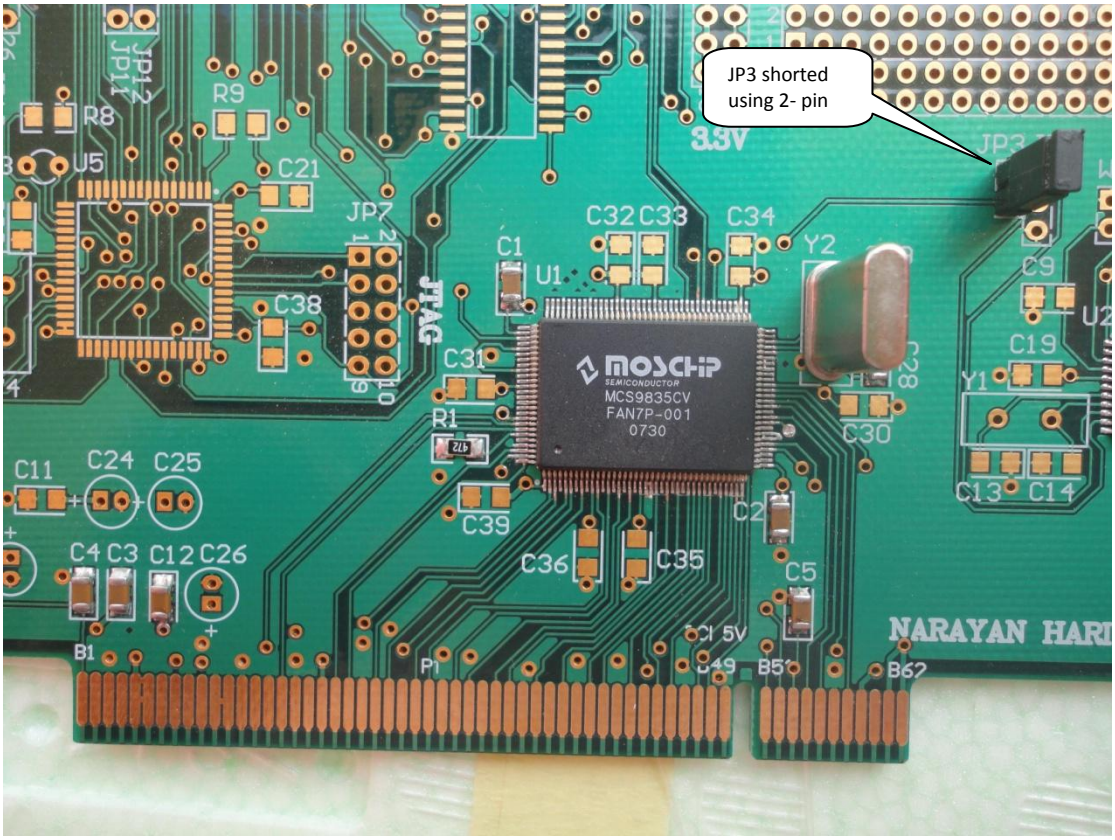


Figure 8.1 MCS9835CV chip soldered.

2. Insert the PCI card into the PCI slot checking the 5V notch on the card. Push in the card firmly so that it cannot vibrate easily. Be sure to maintain distance from other non-insulated cables, so that they do not touch any part of the card. The figure 8.3 shows the card inserted into the PCI slot of the PC. **During the entire procedure of step 3 the PC must be power off.**

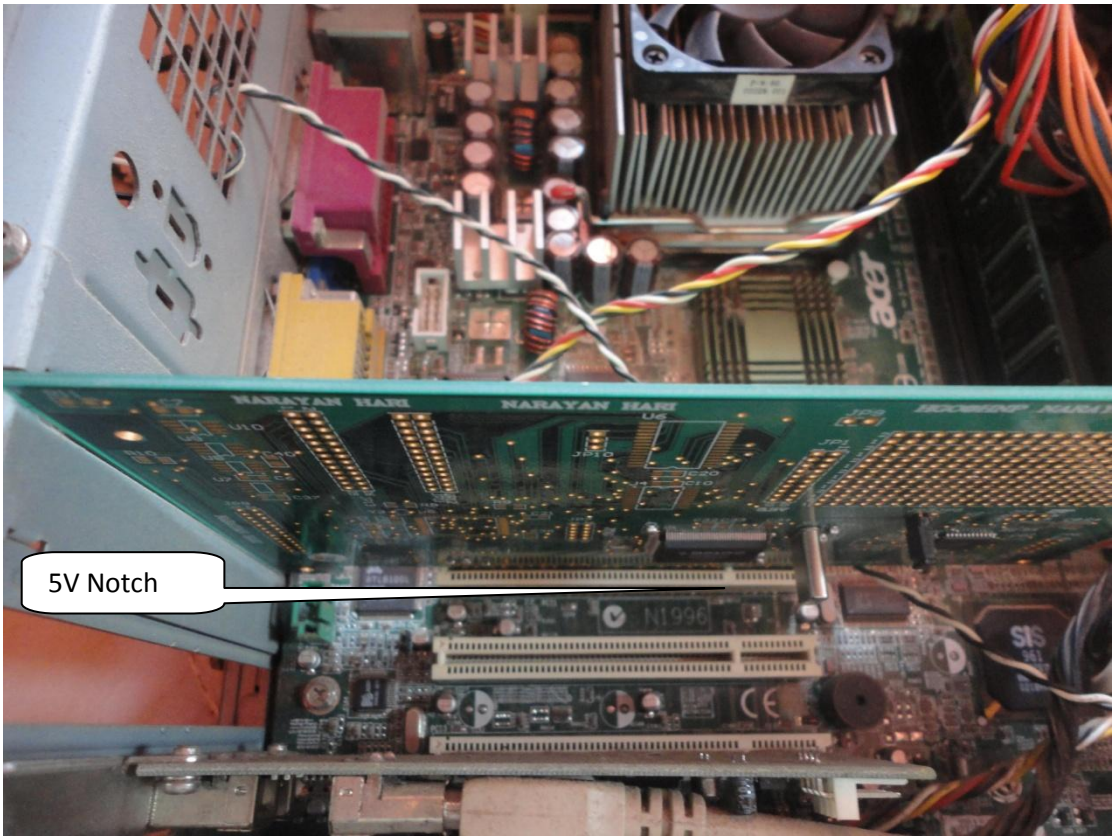


Figure 8.2 PCI card inserted into PCI slot

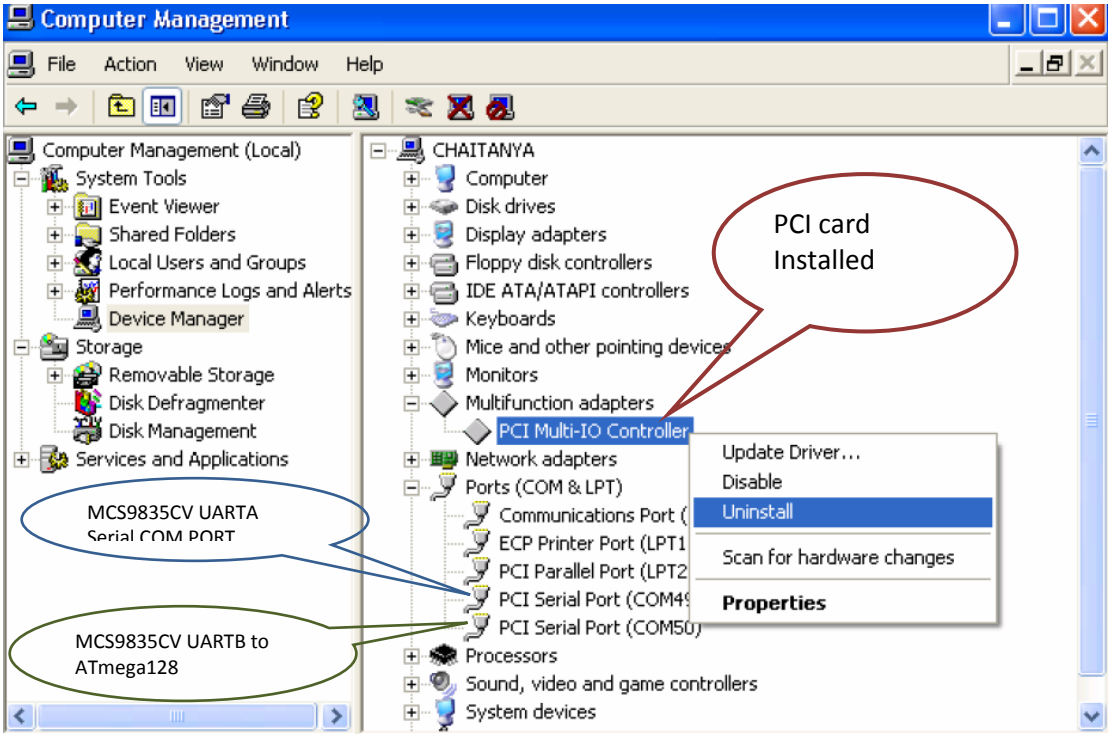


Figure 8.3 View of PCI card installed.

3. Now start the PC and open the Device Manager as shown in figure 8.3 and see to it that all the items appear same as in this figure if not repeat the steps to install the device drivers. Now insert the 2-pin jumper on header JP3, this shorts the pin 109 RXA i.e a serial receive pin of UARTA and pin 105 TXA i.e. a serial transmit pin of UARTA.
4. Open the windows default communication application software Hyper-terminal and set Baud rate 9600 and other protocols 8-N-1 with no hardware flow control. Select the option to echo back the character typed. If having problems with these settings, use Hercules application program from HW-group.com. This is easy to use as multiple characters can be entered and also the HEX format can be inserted. Setup shown in figure 8.4 is used. The characters that are sent must be echo backed.

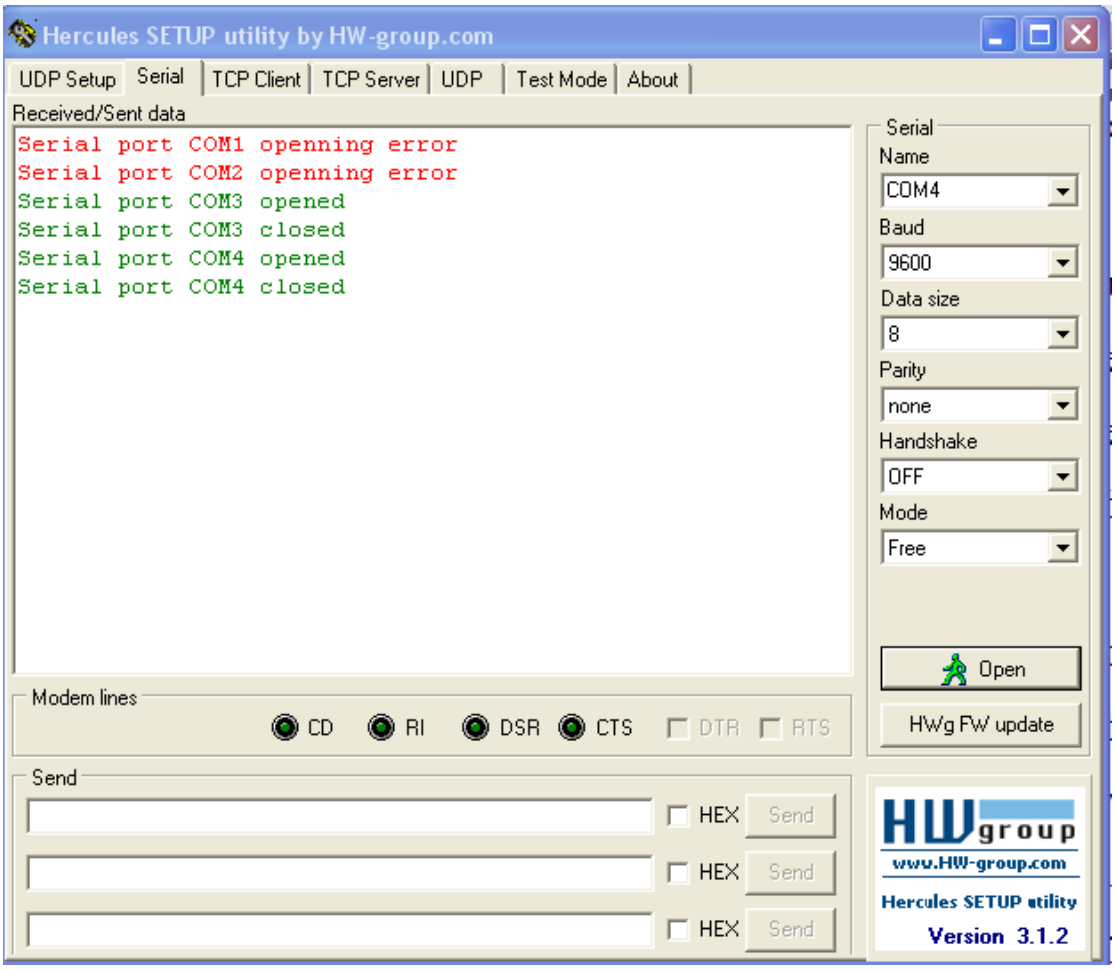


Figure 8.4 Setting of COM port.

Thus the testing of PCI controller subsystem is completed.

Troubleshooting.

If the character sent are not received back then check for the crystal connectivity, also be sure not to route the UART signals under the MCS9835CV. If the design does not work then the PCB routing must be checked for proper routing of PCI signals, the PCI signaling rules may not have been followed. The best part is to send the design to the technical support executive of the MosChip Company. If the design does not work in 2 layers PCB, then re-route the same for 4 layers PCB with the layers stack as pointed in section 7.2 of chapter 7.

8.2 Testing of JTAG-ICE subsystem.

After successfully testing the PCI controller subsystem, the next step is to test the initial working of firmware of the JTAG-ICE flashed through the bootloader developed in section 7.3 of chapter 7. Before flashing the JTAG-ICE firmware the bootloader must be tested using the AVRProg commands given in table in section 7.3 of chapter 7. The serial port UARTA of the PCI card is used to communicate with the Amega16L the same that is demonstrated in section 9.2. The figure 8.6 shows the response of the bootloader in the ATmega16L.

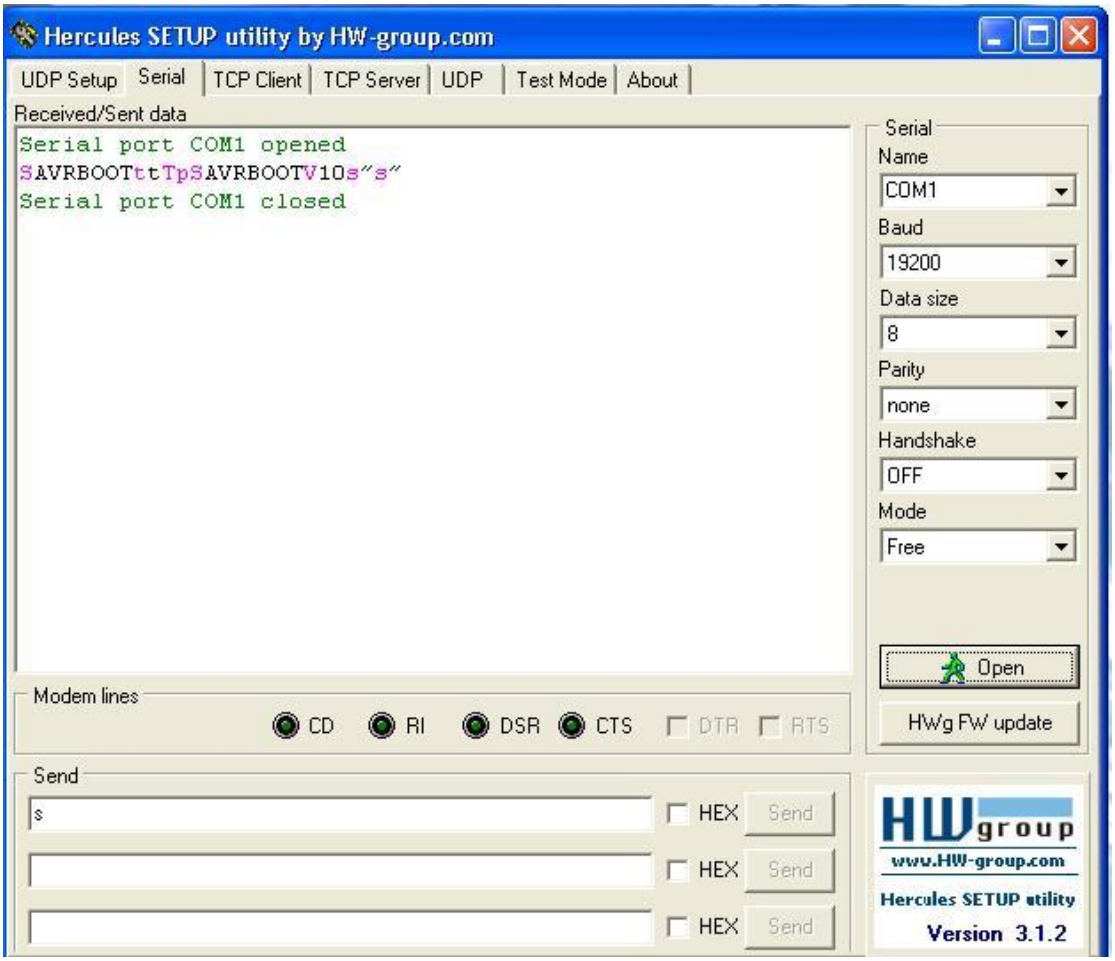


Figure 8.5 AVRProg response from ATmega16L

The bootloader JtagISP.hex can be flashed by any AVR programmer as described in the operating manual section 9.2 of chapter 9, here it is flashed using ISP with AVR Dragon.

Follow the steps to test the JTAG –ICE protocol as per the Atmel application note AVR 060 JTAG ICE Communication Protocol.

1. After flashing the bootloader, momentarily hold the 2-pin jumper on header W2 so that the ATmega16L will be reset. Now the JTAG LED blinks and it's the signal that the bootloader is ready to take commands. Put the 2-pin jumper on W1 header so that the ATmega16L goes to the ISP mode.
2. In the AVR Studio click on the Tools menu and then on AVR Prog, here the bootloader will communicated with AVR Prog. Now select the file ugrade.edn from folder C:\Program Files\Atmel\AVR Tools\JTAGICE\

and press the program button. Here the JTAG-ICE is programmed inside the ATmega16L.

3. Just reset the ATmega16L and use the JTAG-ICE. The JTAG signal pins from ATmega16L are connected to the JTAG port of ATmega128; also the same signals are taken at JP7. This is the facility to use the external target AVR other than the on-board ATmega128, only if ATmega128 is not soldered. At this point we do not solder the on-board ATmega128, but use an external target AVR ATmega32 in a general purpose board. Be sure not to have Vcc power contention with external board from JP7 pin 2, 5Vcc.
4. Open the Hercules setup utility and set the Baud at 19200 and protocol as 8-N-1 with hardware control off. Give the commands of table 8.1 and see the response in figure. Give command and after that Sync_CRC/EOP = 0X20, 0X20. Here the response will be in hex format that has to be converted to its ASCII format for better clarity, some of the commands will not show results, but it is not like that, the display is not in ASCII. Here the testing of JTAG-ICE communication protocol is done with AVR Application Note 060.

Hex	**Command**	**SEQUENCE**	**ASCII response TESTED**
0x20	Get Synch	**[Resp_OK]**	**A**
0x53	Get Sign On	[Sync_CRC/EOP] **[Resp_OK] [“AVRNOCD”] [Resp_OK]**	AAVRNOCDA
0x64	Get Debug Info	[Sync_CRC/EOP] **[Resp_OK] [0x00] [Resp_OK]**	AA

Table 8.1 JTAG-ICE commands

5. Open the AVR Studio and select the JTAG-ICE platform and the serial PORT given to UARTA of MCS9835CV and click connect button as shown in the figure 8.6.

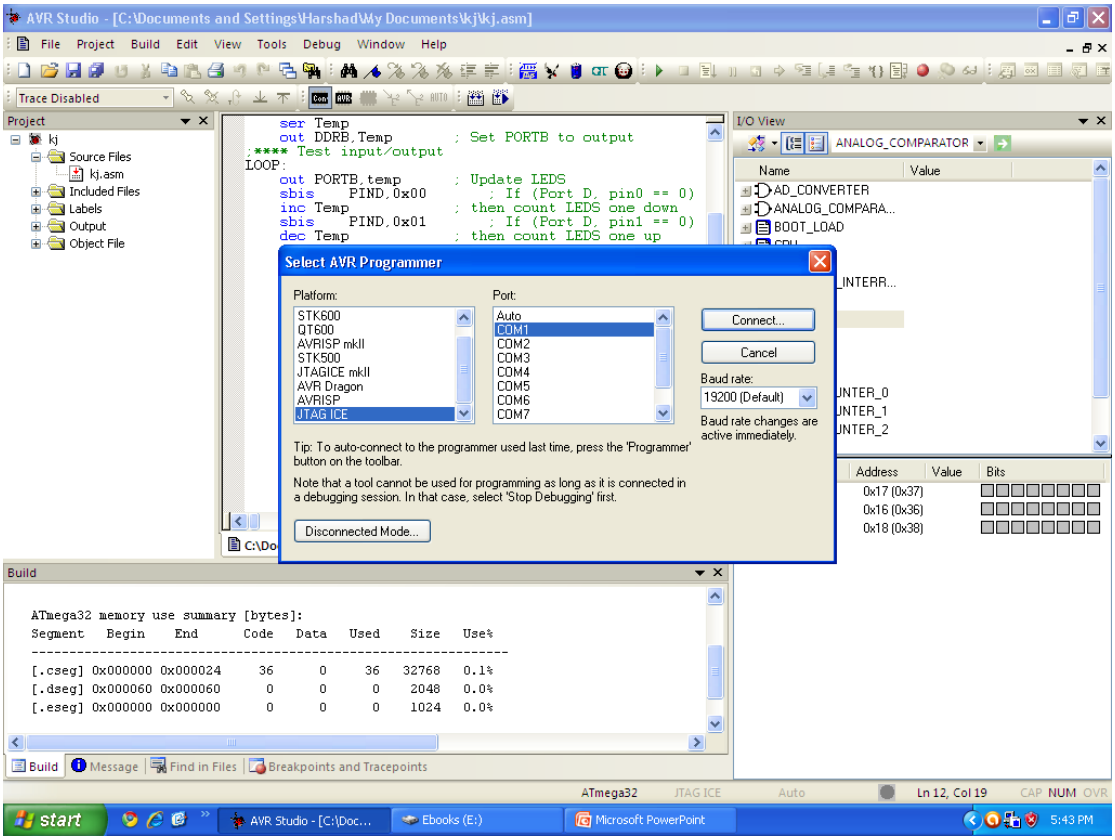


Figure 8.6\Selection of JTAG-ICE.

Select the target as ATmega32 as per the step 1 as shown in figure 8.7.

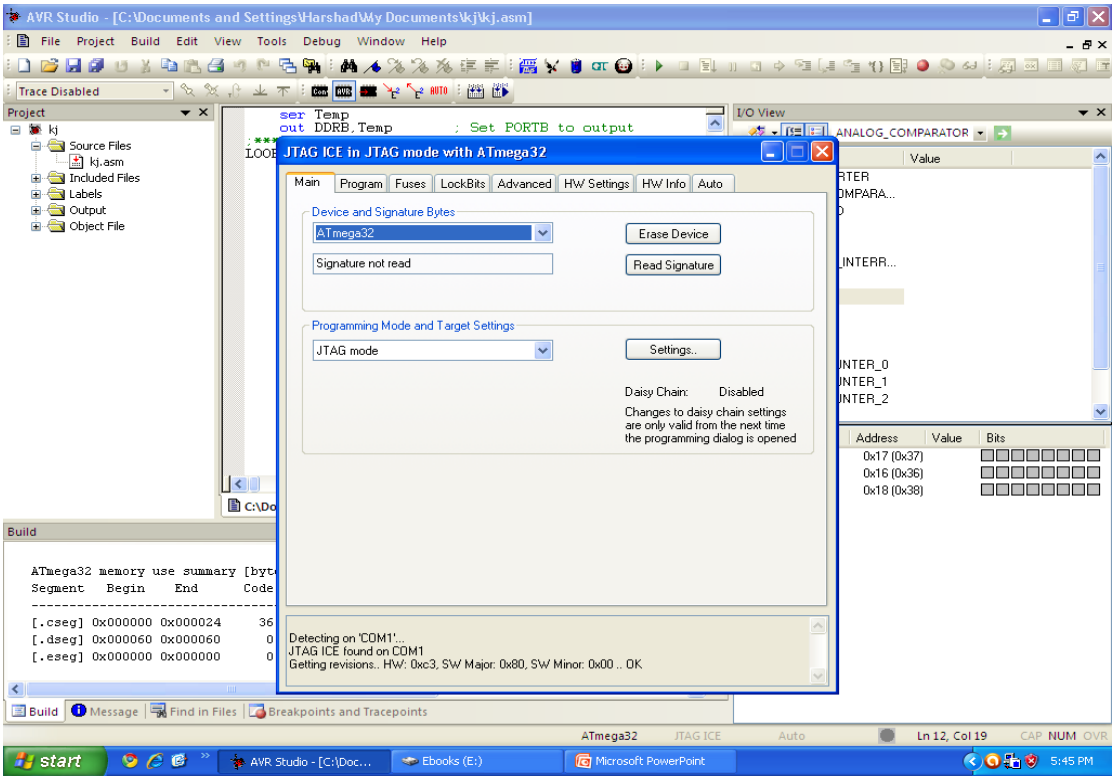


Figure 8.7 Select the Target.

6. Check the power on the target supply by clicking on the HW settings tab as shown in the figure 8.8.

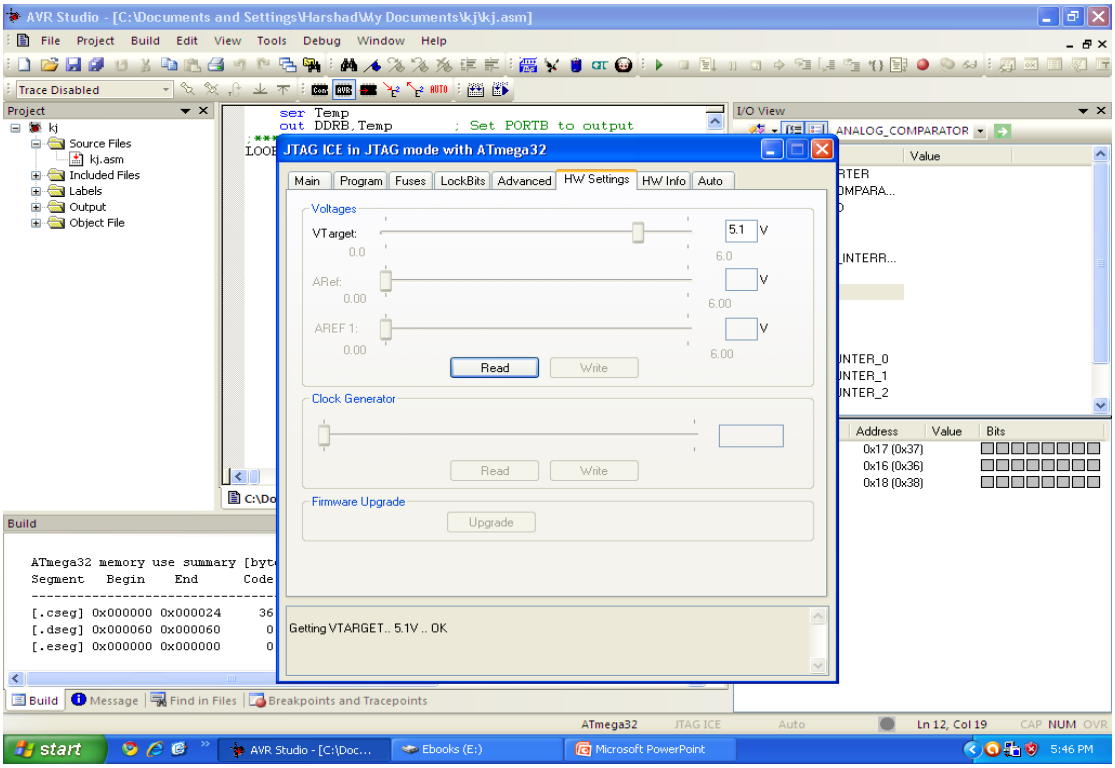


Figure 8.8 Reading the Target Voltage.

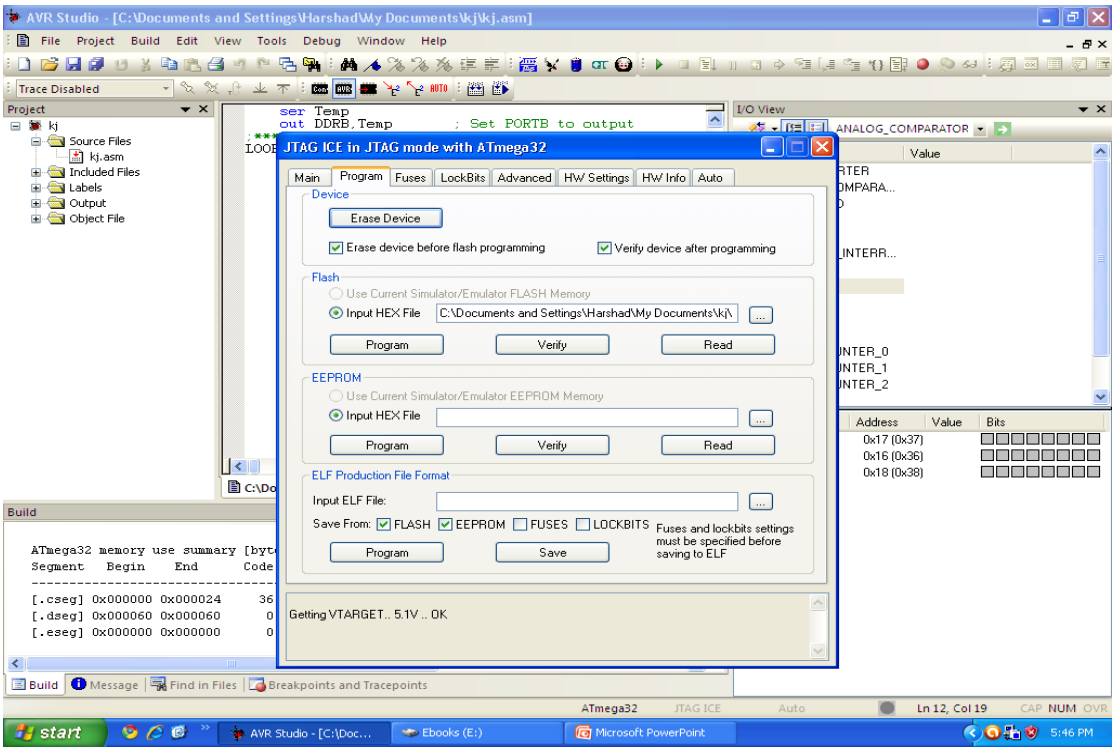


Figure 8.9 Program mode of JTAG-ICE.

7. To program any hex file into the target just go to the Program tab and input the path of hex file and press Program button. The figure 8.9 shows the screenshot of the program mode of JTAG-ICE.

The above step confirms that the JTAG-ICE firmware that was flashed was correct. Next step is test the debugger section

The detail discussion of using the JTAG debugger for the target ATmega32 is as follows:

1. Just follow the above steps for using the JTAG interface and confirm the working of the AVR JTAG-ICE. The demonstration of debugging is shown using the sample program of the STK500 kit that is applicable to our target ATmega32 board. Here no external circuit is to be connected.
2. Open the project in AVR Studio or create a new project while selecting target chip as ATmega32 and click Build and run in the Build main menu. Now single step each instruction and examine the changes in the respective register, I/O and RAM. The figure 8.10 shows the debugging of a simple program. Just examine the affecting registers.

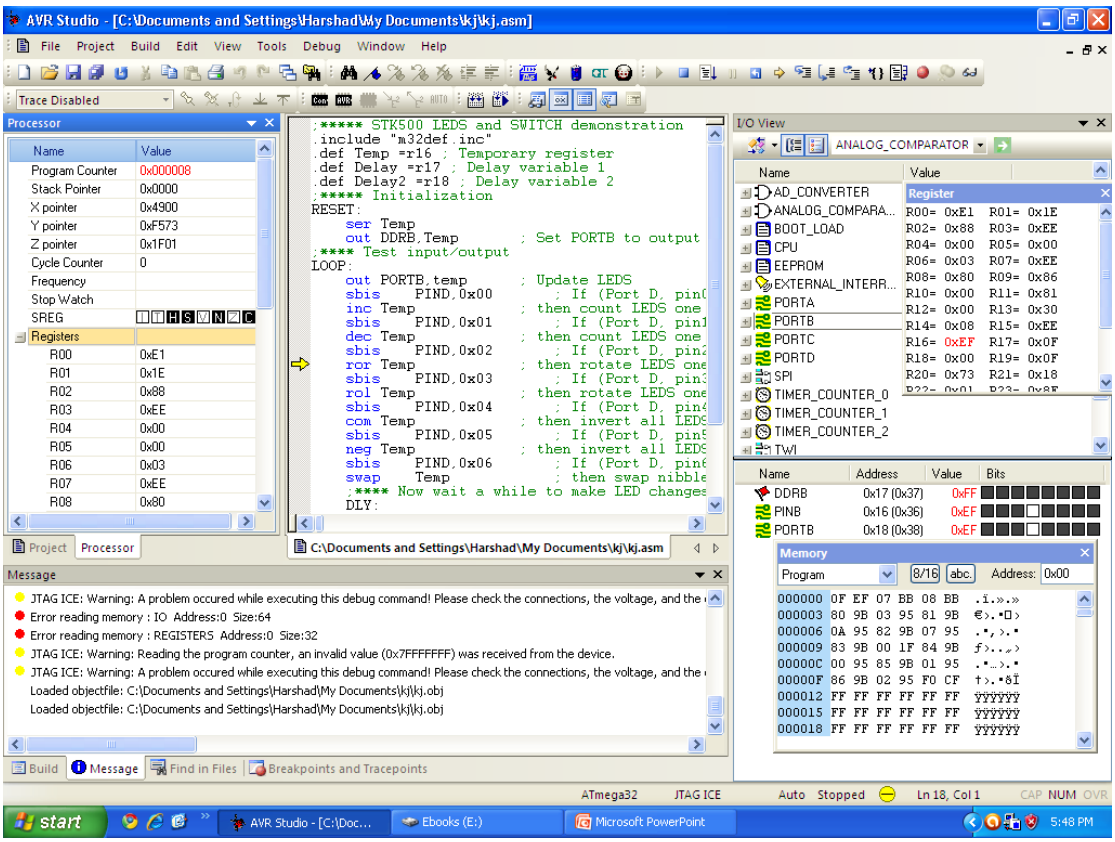


Figure 8.10 JTAG debugging window.

Detail description on how to debug in AVR Studio is in section Operating manual of chapter 9. After successful testing of these subsystems the target ATmega128 along with SRAM 62256, 24C512 and the external card connecting to header JP8 ext2 will soldered and connected.

The JTAG debugger/programmer developed here supports the following AVRs:

- ATmega128, ATmega128L, AT90CAN128
- ATmega16, ATmega16L
- ATmega162, ATmega162L, ATmega162V
- ATmega165, ATmega165V
- ATmega169, ATmega169L, ATmega169V
- ATmega32, ATmega32L
- ATmega323, ATmega323L
- ATmega64, ATmega64L

8.3 Testing of Modules.

For testing each of the modules developed in section 7.5 an UART interface may be supported for feedback of the results. The above two sections described the testing of PCI controller and JTAG-ICE and this section describes the testing of programs for target ATmega128 soldered on the PCI card.

8.3.1 Basic Input/Output Module with external interrupts.

To test the basic I/O connect the external board, the circuit diagram is shown in figure 7.20 of chapter 7. The actual board PCB pattern is shown in figure 8.11.

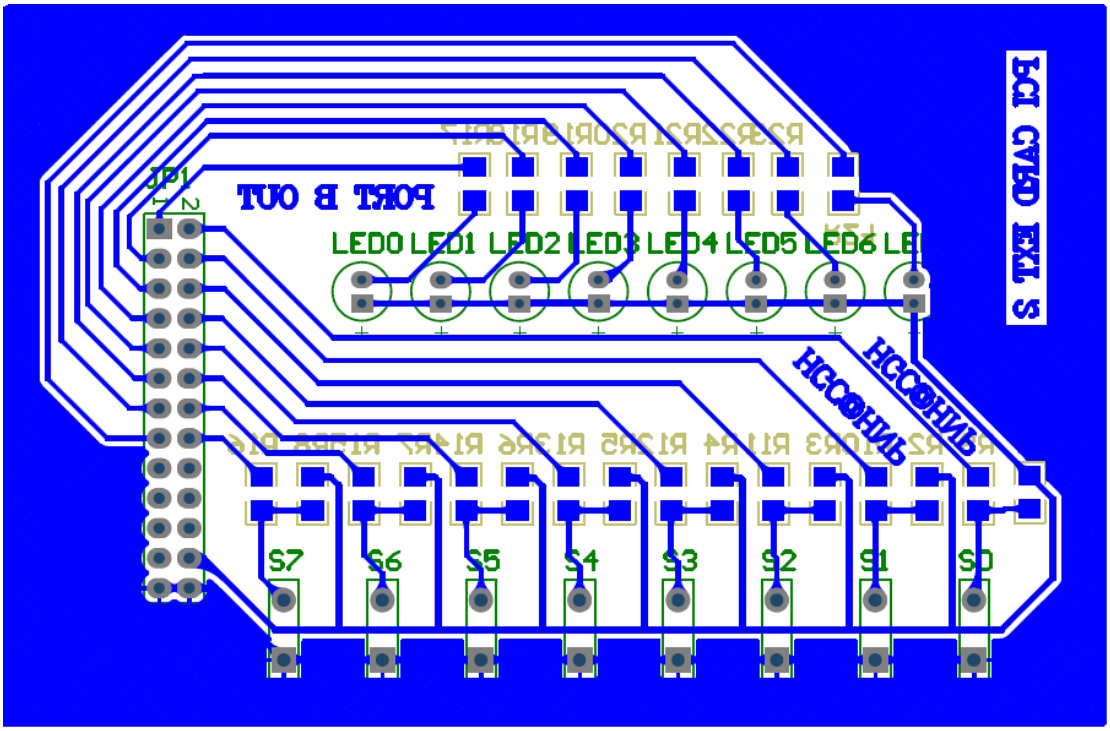


Figure 8.11 External I/O card

The TUT_14.hex file and TUT_14.cof file is generated by Flowcode and use the TUT_14.cof in the AVR studio for simulation. As this code is generated outside the AVR Studio, it has to be imported to AVR Studio. After running the code, it is observed that the LEDs are running from left to right.

8.3.2 PWM interface control via UART0 module.

The example shows the dimming of the LED4 on the external I/O card using the PWM phase correct method of ATmega128 at its pin OC0, PB4 and the change in dimming can be done through UART0 of ATmega128. Connecting them to the UARTB of the PCI card, hex data can be send to vary the dimming effect of LED4. The AVR project for this module is developed using AVR assembler in AVR Studio

To test this code just open the COM port related to UARTB of PCI card and Run the program in the AVR Studio. Now input the hex characters to observe the change in brightness of LED4.

8.3.3 I2C Interface EEPROM module.

The AT24C512 stores 64Kbytes. The sample program for storing data in U7 is demonstrated. Data is stored from address 11 to 20 of U7 and retrieved back using UART0 of ATmega128. The firmware is developed in BASCOM-AVR using BASIC lauguage for AVR microcontroller. Try opening the current

project from the AVR studio and select the M128I2CEPROM.OBJ file and load it. The Studio will automatically create the project and open the project after selecting the JTAG-ICE platform and target as ATmega128. The COM port displays the result of storing data into the AT24C512. A detail regarding using BASCOM-AVR for debugging is described in section operating manual of chapter 9.

8.3.4 Parallel interface SRAM module.

The XMEM interface of ATmega128 can interface to external SRAM, EEPROM, ADC. DAC other bus supported peripherals. Here SRAM is interface externally using 74AC573 latch so that the ATmega128 supports 32Kbytes of SRAM along with 4Kbytes of internal SRAM. The project of assembly program listing is given in section 7.5.4 of chapter 7. Running the project in single step mode it is noticed that writing/reading data to internal SRAM is a simple task, but to access utmost care must be taken to refer the external address. The configuration of figure 8 must be taken into consideration before using XMEM. The AVR assembly language has instruction well capable to understand which memory to address due to its RISC Havard architecture. The table 8.2 gives proper understanding for the access of address in ATmega128.

Actual Memory in HEX	Memory of Internal SRAM	Memory of External SRAM
0x204	0x204	-NA-
0x8204	-NA-	0x204
0x7204	-NA-	0x7204

Table 8.2 Example of addressing SRAM using XMEM

8.3.5 ADC module via UART0

For testing the ADC of ATmega128 simple potentiometer is attached to pin PF0, ADC0 and AVref is connected to 5Vcc so that 0 to +5V dc can be measured. The measured result is send to the UART0 of ATmega128, which is connected to UARTB of MCS9835CV and is viewed on the Hercules serial setup. The coding of this module is available in sample exercise. The coding is done for single-ended mode operation. The program is developed in BASCOM-AVR compiler. The values of spot voltage on pin PF0 is shown

continuously on the terminal. The same can be verified using DMM at pin PF0, the results are perfect.

8.3.6 1-wire iButton interface.

The iButton device from Maxim-Dallas DS1990A is worked on and an article is also published in the Electronics For You magazine in November 2010. It features the 1 – wire technology that do not require power for its working but it lives on the parasitic power when it is accessed as stated by the company. Simplest form of 1-wire device DS1990A is a stainless steel button in which there is a computer chip on which 64-bit permanent laser printed ROM. The address of the DS1990A is used for identification as it is unique and is displayed on the serial terminal. Just single pull-up is required at pin 25 PD0 that acts as external interrupt. There is a lot of support for 1-wire devices on the website. The program for this module is developed in BASCOM-AVR using its 1 wire library and can be debugged in JTAG-ICE as discussed in the above modules.

8.3.7 AVR Studio GUI plugin.

The AVR GUI plugin is developed using AVR Studio SDK, which is a Dynamic Link Library DLL file. The coding of the plugin is done in purely VC++, but it is imported to VC++.NET using Microsoft Visual Studio 2008 and compiled for the DLL. It is found in the directory as shown in figure 8.13.

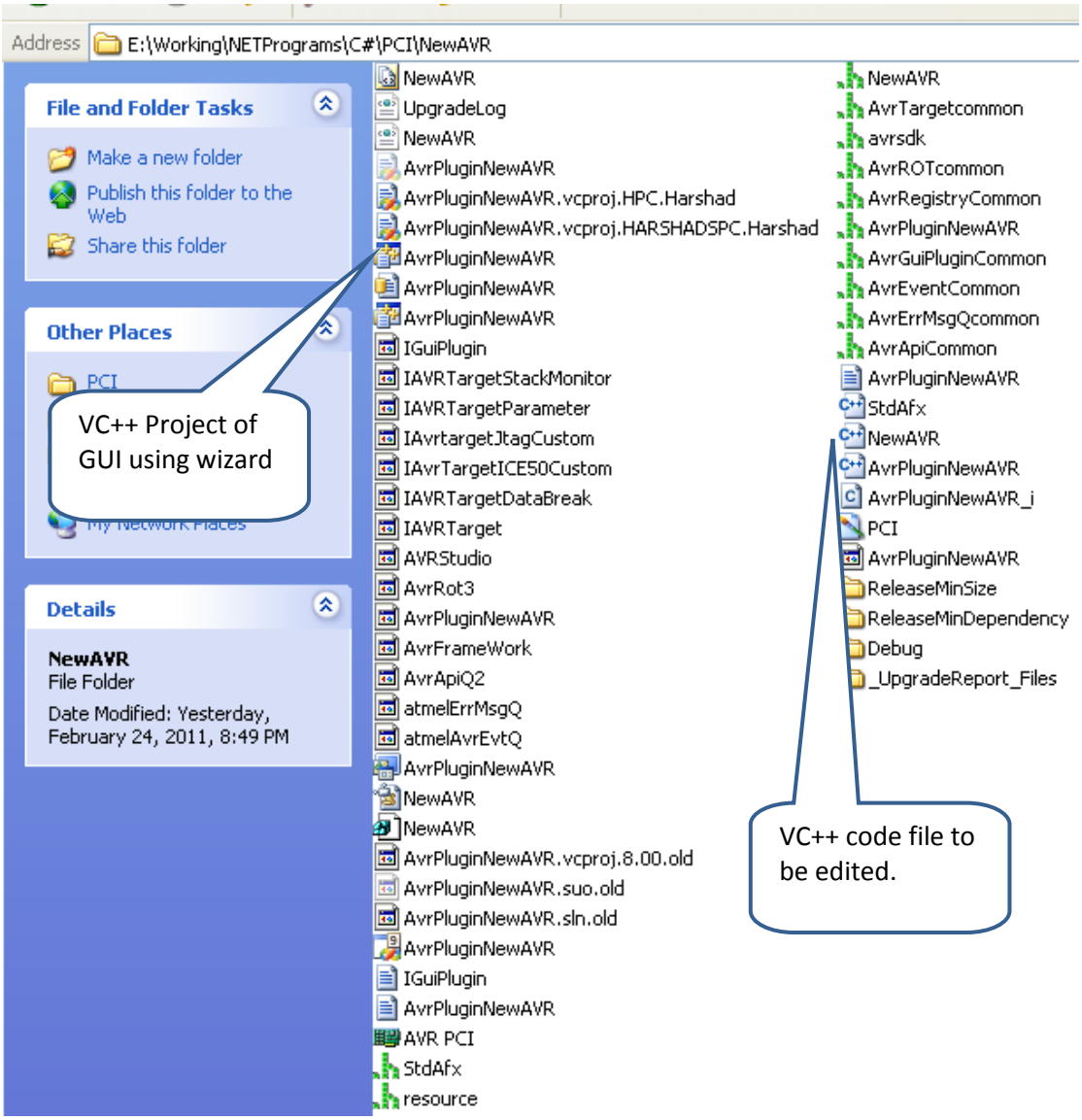


Figure 8.12 Project folder of GUI Plugin

The DLL file to be is to be registered in windows to use it using command regsvr32 in the run command line of start button of Windows OS. After registering the DLL file the figure 8.13 shows the GUI sample of calculator opened and loaded.

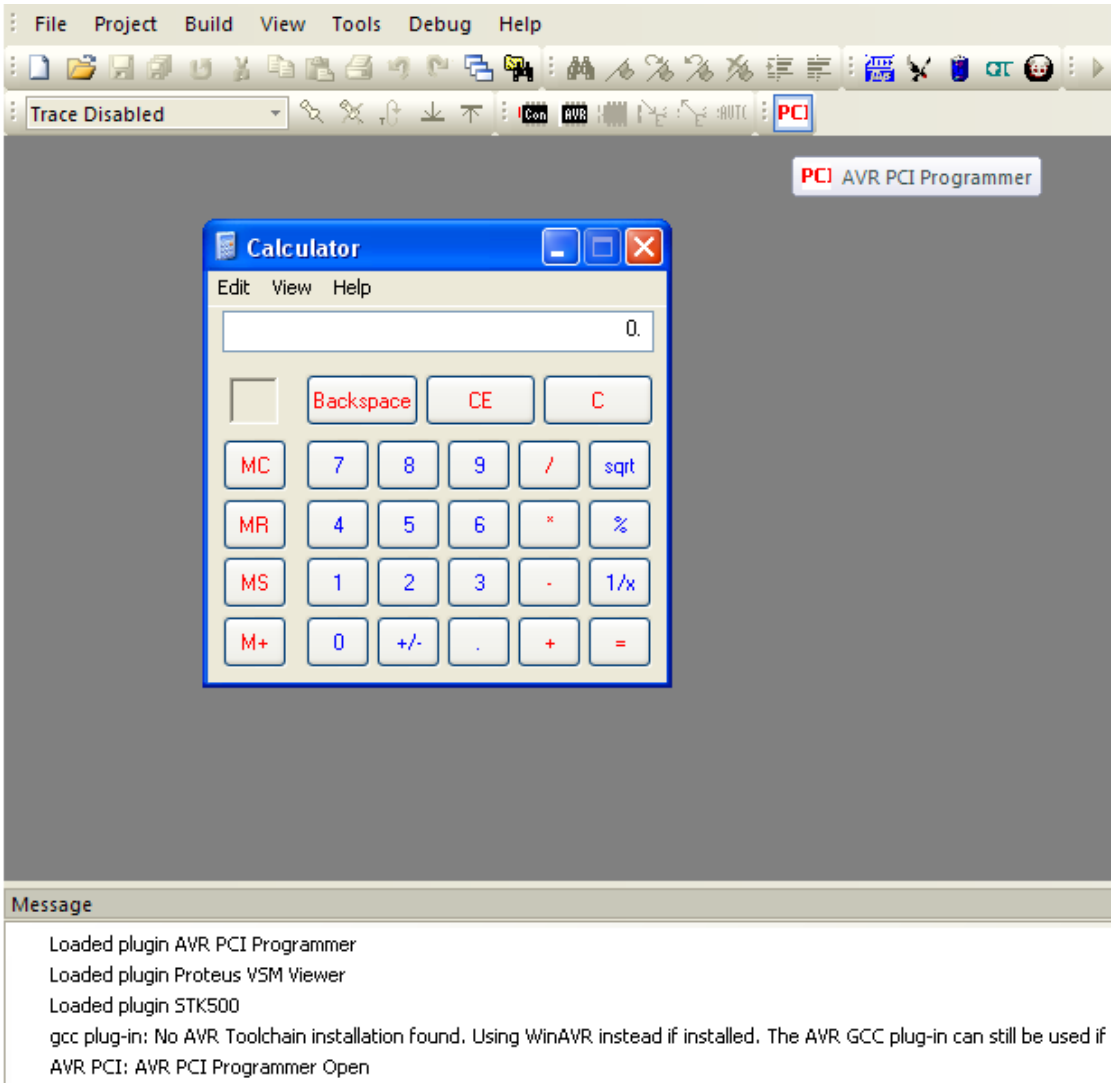


Figure 8.13 Sample GUI Open and loaded

The figure 8.14 shows the plugin added to the tools menu.

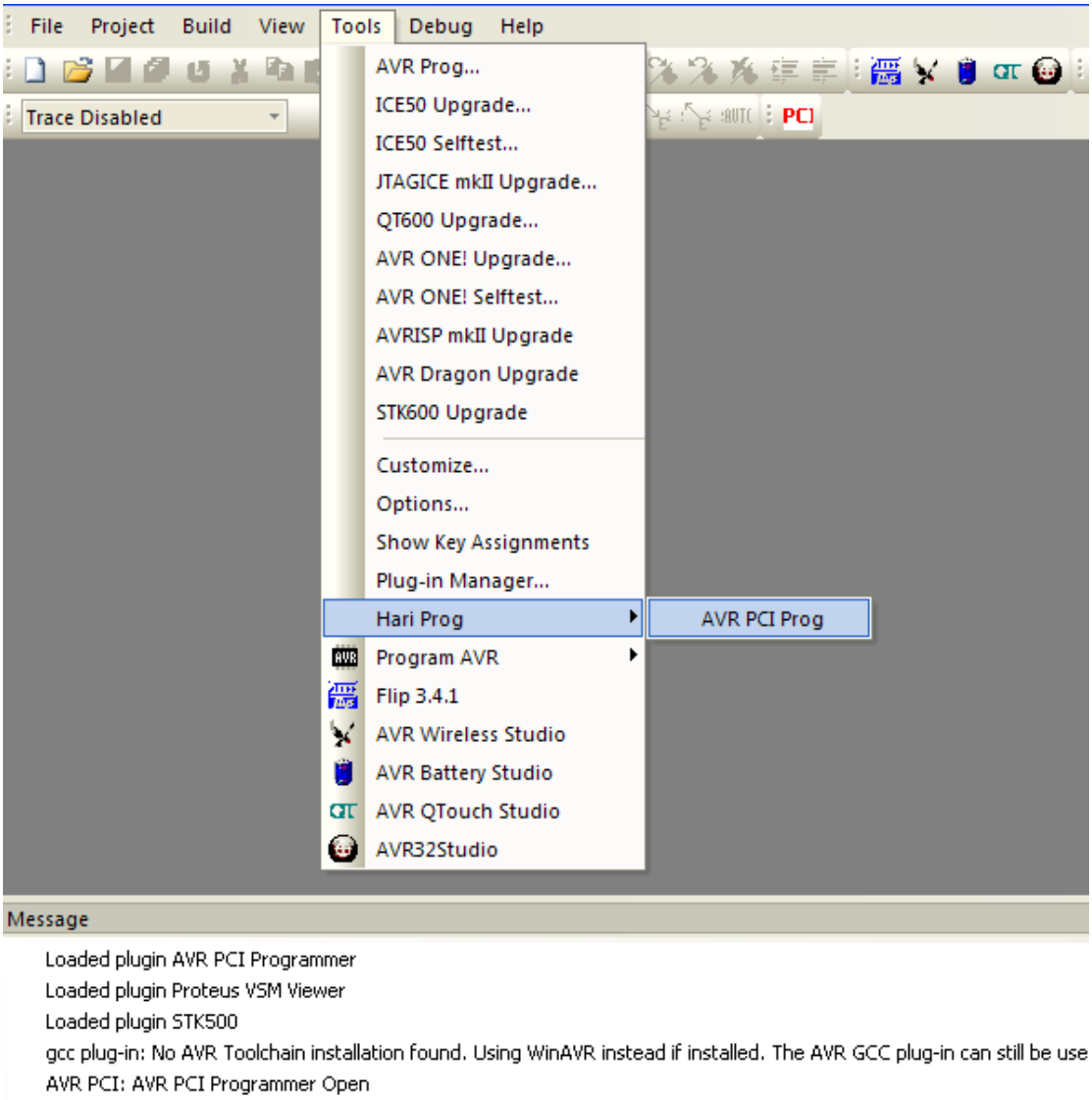


Figure 8.14 Plugin added to Tools Menu

8.4 System Integration and Testing.

System integration is done after executing and working on with subsystems in the testing phase, the simultaneously taking each of the subsystems and combining to give the required result. After testing the PCI controller subsystem and the JTAG ATmega16 section along with target ATmega128 performs the role of system integration.

Testing the functioning of MCS9835CV, JTAG-ICE and target ATmega128 with peripherals and using target ATmega128 for interfacing GSM modem and serial memory and µC-OS II RTOS that is specially designed for AVR ATmega128 contributes to system integration firmly. The developed PCI card is also ready for loading RTOS available from various companies.

8.5 Discussion of Testing Strategy.

The use of Development tools such as compiler toolchain, debugger, and prototype platform for the target microcontroller depend much on the testing strategy. Because the toolchain including the IDE must support the debugging frontend application software properly, maintain communication between the device drivers of the host PC also.

The testing strategy tells that build the hardware so as to accomplish the requirement and check it using DMM and DSO. Program the firmware and check the resultant output is up to the mark. The main point behind the testing strategy is to divide and work. Just divide the program into specific parts and after successful results of each of the part just work on the next related part, so the same strategy is adopted for 1 wire DS1990A device. The implementation of developing a proper bootloader is a little tedious job, but going step by step the concept is realized.

Chapter 9

Installation and Maintenance

Chapter 9 Installation and Maintenance.

This chapter it important for new newbie user, as it is like a start up guide for using the PCI card. The section 9.2 gives all the information to start playing with the PCI card. Before doing anything with PCI card first of all hardware and software requirements must be known, otherwise it is difficult to quickly have hands on the card.

9.1 Hardware and Software System requirements.

System requirements for AVR studio.

Recommended hardware:

- Intel Pentium 200MHz processor or equivalent.
- 1024x768 screen (minimum 800x600 screen).
- 256 MB memory.
- 100 MB free hard disk space

Recommended Software:

- Windows NT/2000/XP/XP x64/ VISTA/7 x32 /7 x64.
- Internet Explorer 6.0 or later (Latest version is recommended).

Windows 95 is no longer supported by AVR Studio. The most recent version that supported Windows 95 was AVR Studio 4.12 SP3. Windows 98 is no longer supported by AVR Studio. The most recent version that supported Windows 98 was AVR Studio 4.16 SP1.

System Requirement for AVR SDK Software:

- AVR Studio 4.12 or later.
- Microsoft Visual Studio 6 or Microsoft Visual Studio .NET

Any tool capable of creating a COM object can in theory be used, but the SDK assumes use of Microsoft development tools for C++.

System Requirement for WinAVR/AVRtoolchain

- AVR Studio 4.10 and above.

System Requirement for PCI card.

System with PCI slot on its motherboard and with any of the following operating systems

- Windows 32bit OS – 2000 / XP / 2003 Server / Vista.
- Windows 64bit OS – XP /2003 Server / Vista.
- Linux.

- DOS 6.22.

9.2 Operating Manual

The operating manual gives the path for the first AVR program to be executed on target AVR ATmega128 of the PCI card. First of all it is needed to insert the PCI card looking at the proper notch for 5V card in the PCI slot of the PC. The figure 8.3 shows the PCI card inserted inside the PCI slot. The PCI card must be inserted only when the PC is shut down.

Installing the MCS9835CV Driver.

1. After inserting the PCI card, start the PC and figure will appear after the boot process is complete.

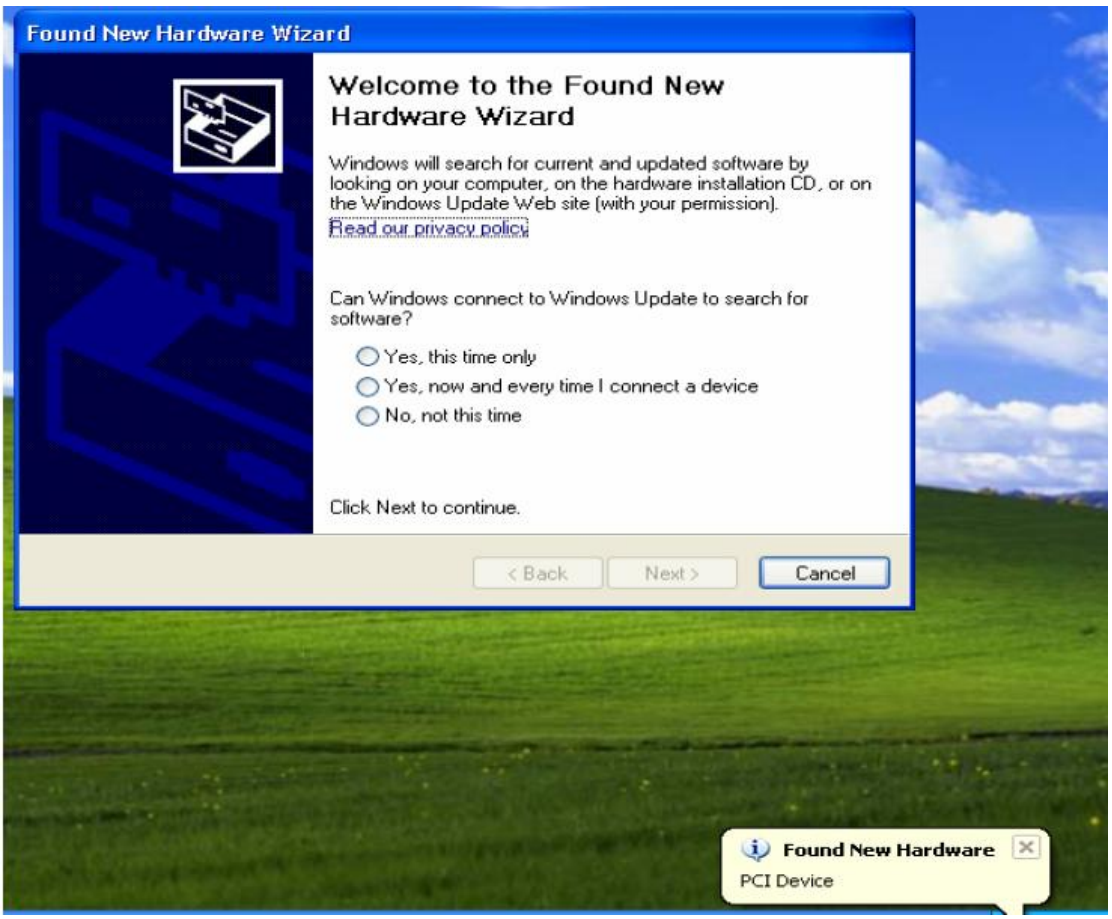


Figure 9.1 Device Found Wizard

2. Click on third option and click Next and specify the path of the driver and complete the installation process, now the notification area will show “New Hardware is installed and ready to use”. Restart the Computer.

3. Verify the installed drivers by opening the Device Manager as shown in the figure .

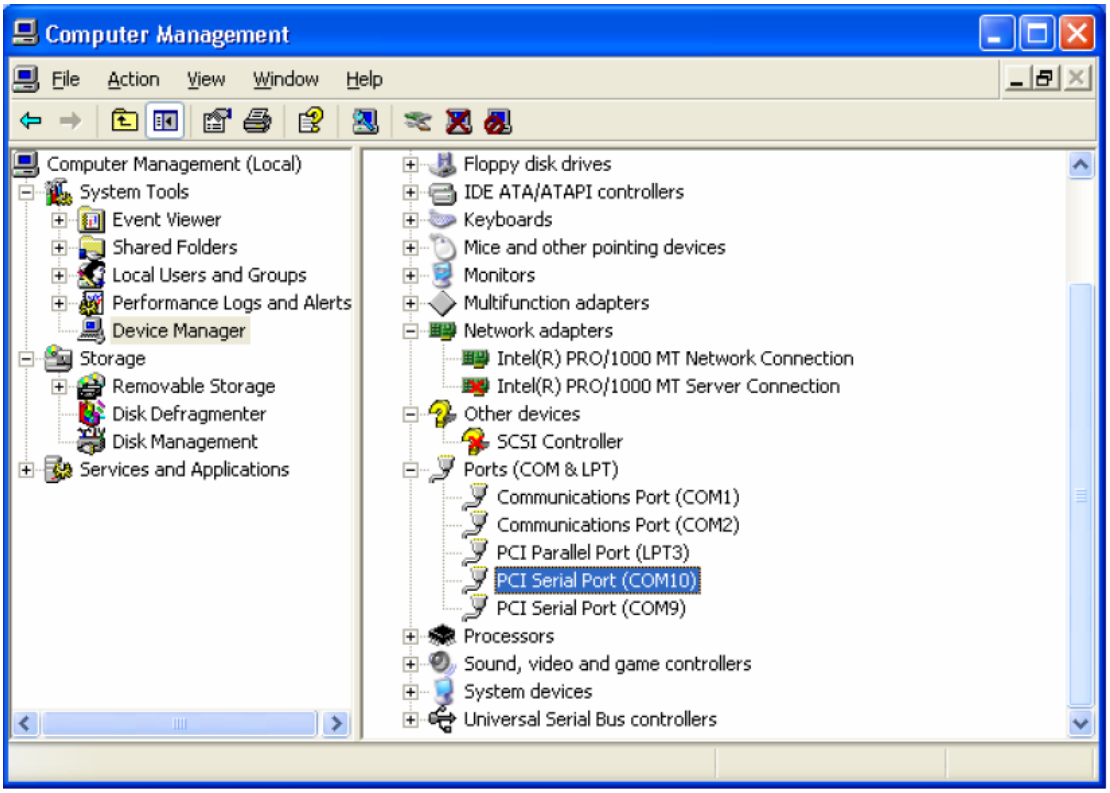


Figure 9.2 Windows Device Manager

4. The serial properties can be viewed by right clicking the desired port.

Flashing the bootloader on the ATmega16L.

1. Attach the AVR Dragon on header JP4 for ISP programming and insert 2 – pin jumper on W1.
2. Start the AVR Dragon in programming mode select ATmega16 and ISP mode and verify the signature bytes. On the program Tab select the JtagISP.hex file click the Program button. See figure.
3. Give restart to ATmega16L by inserting 2 – pin jumper on W2 momentarily, and then the JTAG LED will start flashing. Remove the jumper from W1. The ATmega16L bootloader is ready to be communicated with AVRProg tool in AVR Studio. See figure.
4. Go to Tools menu in AVR Studio and start AVRProg, and select file “C:\Program Files\Atmel\AVR Tools\JTAGICE\Upgrade.ebn”, click the Program button. See figure. The JTAG programmer/debugger is flashed and ready to use.

Programming and debugging target via JTAG.

All the steps to be followed, to use the JTAG programmer /debugger is given in section 8.3 of chapter 8.

Using the target ATmega128.

To use the features available with target ATmega128 on PCI card, the table lists the peripherals and their interfacing connections with ATmega128.

Sr. No.	Peripherals	Headers/Components on PCB	Pins on ATmega128
1	32Kbytes SRAM	JP10 – GND	XMEM interface. Solder the Latch and SRAM.
2	UART with PC	Direct connection	Pin – 2,3 to UARTB of PCI
3	UART External	JP9	Pin – 27,28 (PD2,PD3)
4	I^2C EEPROM	U7, U8, U9, U10	Solder the AT24C512. Pin – 25, 26 (see Table 7.4 of Chap. 7)
5	External I^2C	JP13 – 2, 4. JP12 – 1, 2.	Pin – 25, 26. Pull-ups available on-board.
6	External interrupt and Input/Output	JP8, JP9, JP11, JP12, JP13	PORTB, PORTD, PORTE, PORTF
7	Real Time Counter	Y3(Crystal)	Pin – 18, 19. Solder 32.768KHz.
8	SPI interface	JP8 – 1,3,5,7	Pin – 10, 11, 12, and 13.
9	Power supply (+12V, +5V, +3.3V)	JP1, JP2, JP8, JP13.	Only +5V

Table 9.1 Peripheral Jumper Selection

Debugging BASCOM-AVR project in JTAG debugger.

1. Install BASCOM AVR

Start installation by running setup.exe:

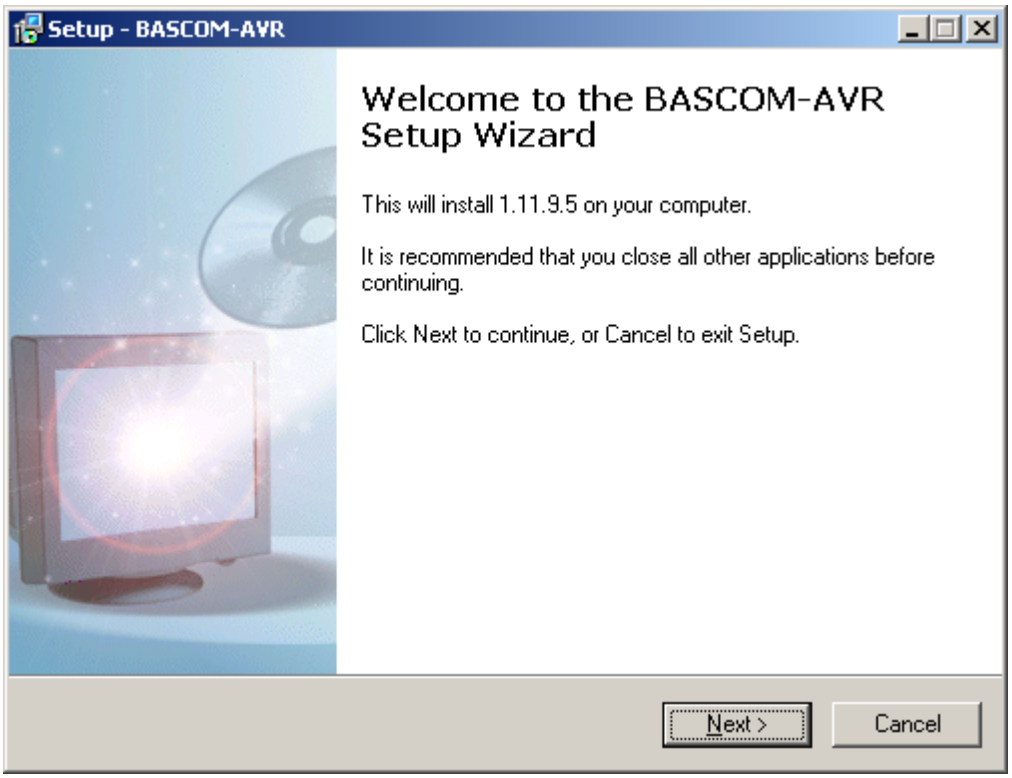


Figure 9.3 BASCOM Wizard Start

Click Next. Accept the license agreement and complete the installation.

2. Open the AVR Studio and in Project menu click on the Open Project.

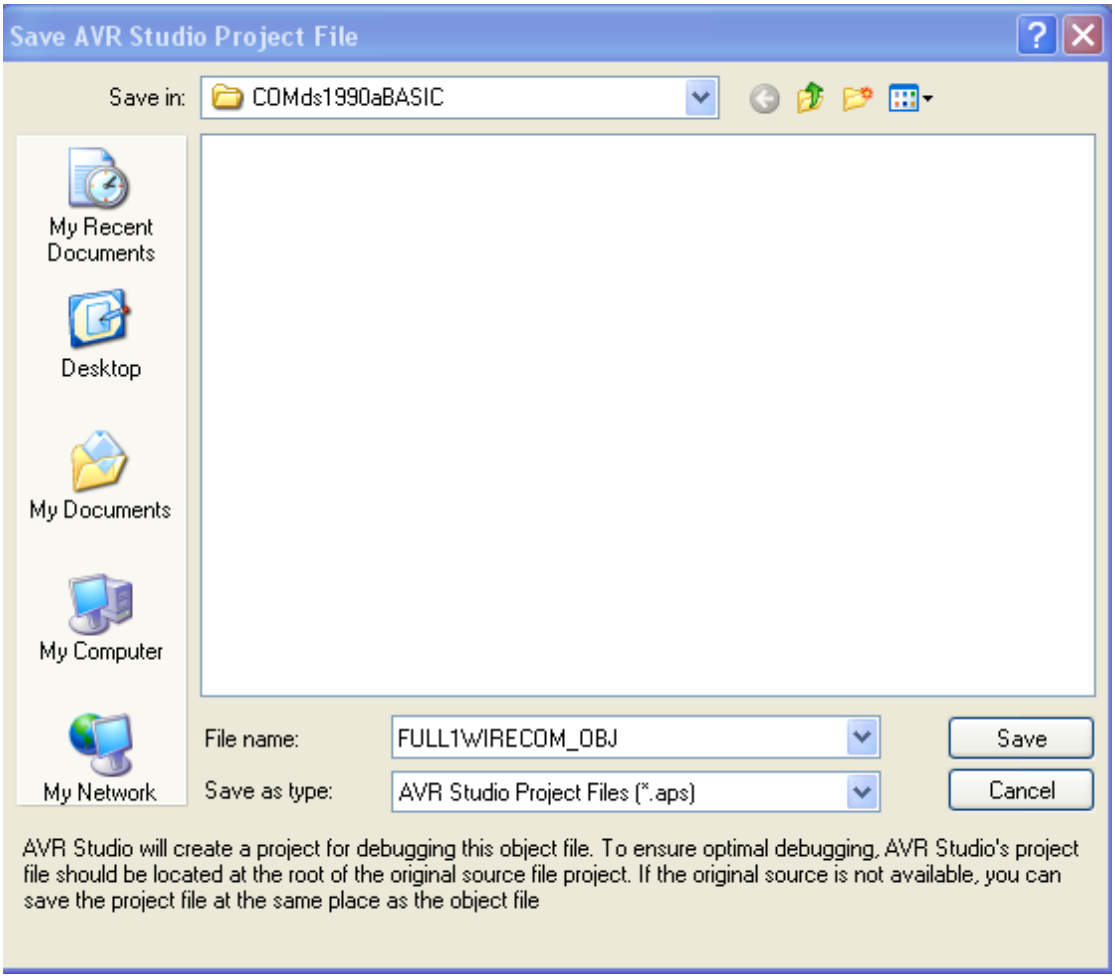


Figure 9.4 OBJ File Selection

3. Select the FULL1WIRECOM.OBJ, use the iButton example. Click Save and then select the Debug platform as JTAG-ICE and device as ATmega128 and click Finish. The screenshot figure will be displayed.

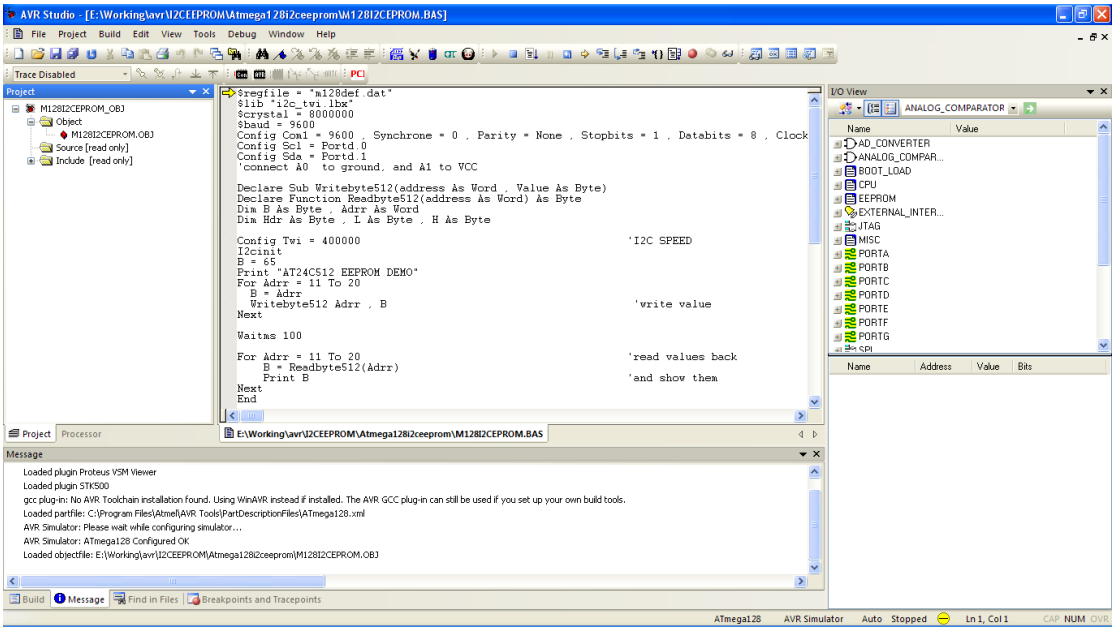


Figure 9.5 Debug Screenshot

4. Now debug the program in normal way as given in the AVR Atmel JTAG-ICE tools help.

9.3 Software

The software that are used in the present research work are:

1. AVR Studio.
2. AVR Studio SDK.
3. WinAVR(AVR – GCC Toolchain)
4. BASCOM – AVR Compiler.
5. Microsoft Visual Studio 2008.

The AVR Studio is an Integrated Development Environment from Atmel Corporation. The IDE supports AVR 8 – Bit microcontrollers for assembling, compiling, simulating and debugging the programs written in assembly and C language. The full on-line help is available from the website www.atmel.com or check in the start menu item Atmel Tools Help as shown in figure

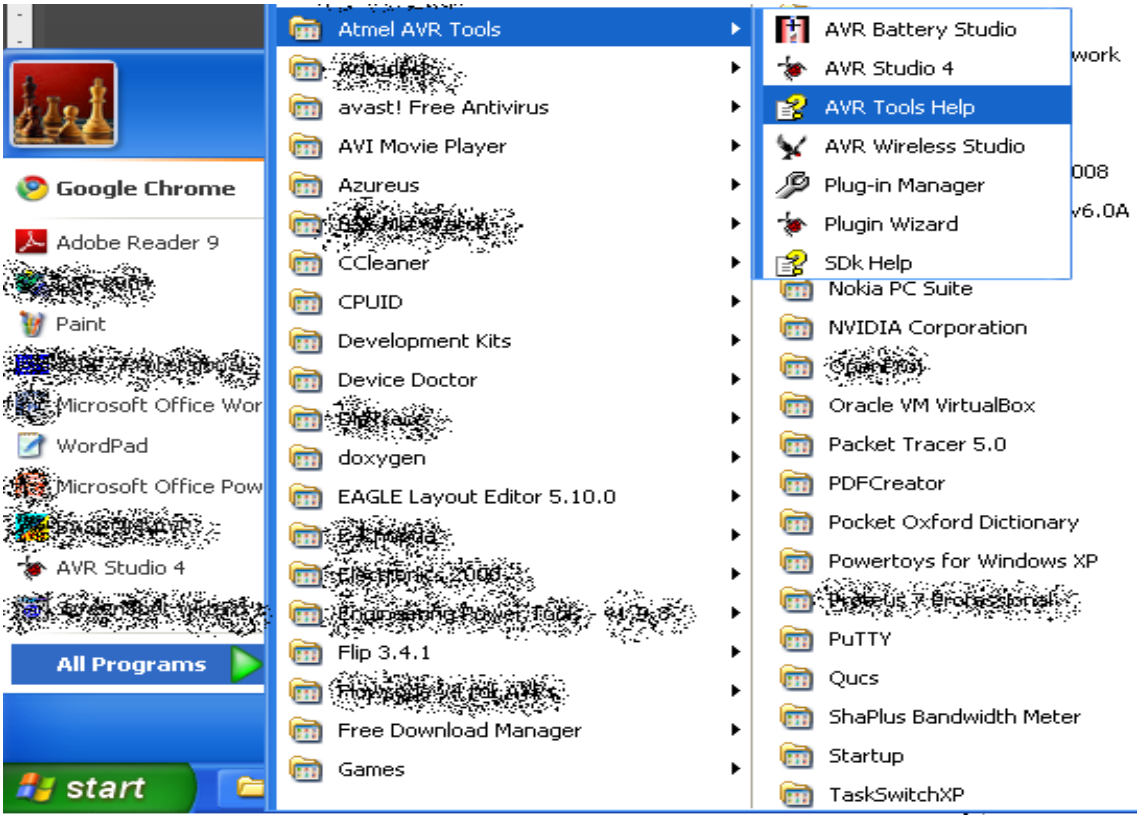


Figure 9.6 Atmel AVR Tools

The WinAVR is free open source setup of development tools for installing and using AVR-GCC utilities with windows. It comes under GNU GCC project for AVR microcontroller which maintains C/C++ compilers and debuggers and also other important utilities for other family of microcontroller and microprocessors. Basically AVR-GCC has utilities that have only command line interface, so WinAVR also includes AVR-GCC plugin that is loaded every time whenever the AVR Studio is opened. In this way the AVR-GCC tools are used automatically and C projects can be build and compiled without bothering about the makefile. WinAVR also includes a programmers' notepad that is not basically used if AVR Studio is used. The WinAVR includes the following main components at present.

1. AVR GNU Binutils 2.19

   Binary utilities for AVR target (including assembler, linker, etc.).
2. AVR GNU Compiler Collection (GCC) 4.3.3
3. C language and C++ language compiler for AVR target. There are
4. caveats for using the C++ compiler. See the installed avr-libc

User Manual in the <InstallDir>\doc directory.

5. avr-libc 1.6.7cvs

   C Standard Library for AVR.

6. AVRDUDE 5.8cvs

   avrdude is an open source programmer software that is user extensible.

7. AVR GNU Debugger (GDB) / Insight 6.8

   GDB is a command-line debugger. Insight is GDB with a GUI!

8. AVaRICE 2.9
9. avarice is a program for interfacing the Atmel JTAG ICE to GDB and users

   can debug their AVR. Use it in conjunction with GDB.

10. SimulAVR 0.9cvs

    simulavr is used in conjunction with GDB to provide AVR simulation.

11. AVR32 GNU Binutils 2.19
12. AVR32 GNU Compiler Collection (GCC) 4.3.2
13. Newlib (for AVR32) 1.16.0
14. AVR32 GNU Debugger (GDB) / Insight 6.7.1
15. Splint 3.1.2
16. SRecord 1.47
17. SRecord is a collection of powerful tools for manipulating EPROM load files.
18. It reads and writes numerous EPROM file formats, and can perform many

    different manipulations.

19. MFile An automatic makefile generator for AVR GCC.
20. Programmers Notepad 2.0.8.718
21. Programming editor and IDE. This editor includes the Scintilla editor component.
22. LibUSB 0.1.12.1 and device drivers
23. This is a USB library that is linked into AVRDUDE and AVaRICE to allow them
24. To connect to the Atmel JTAG ICE mkII and the Atmel AVRISP mkII. Drivers

for these devices are also included.

25. Cygwin DLLs
26. Certain DLLs from the Cygwin project are required for specific included packages. See the Build Notes section for which packages require which DLL.

    NOTE: Not all executables require these Cygwin DLLs.
27. Many native Win32 GNU programs and utilities including make and bash.
28. Tofrodos 1.6 A command-line text file line-ending convertor.
29. A Makefile Template for you to use in your projects.
30. Documentation for the various projects.
31. Source code patches used to build the various projects.

The AVR Studio Software Development Kit was received from the Atmel Corporation by signing the license agreement. The use of this SDK is done only for educational purpose. SDK gives the support to extend the AVR Studio as per user enhancement and use. More tools or data retrieving application interfaces can be developed using SDK. The help of the SDK gives all information regarding using it, and it can be accessed from the link SDK Help shown in figure 9.6.

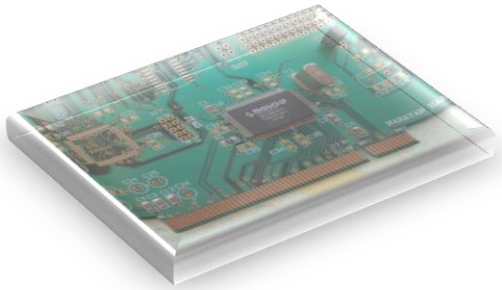
The BASCOM-AVR is an IDE from MCS Electronics that includes compiler which is a BASIC language compiler, a simulator and have library for many peripheral components to be used directly in the program. The language for programming is BASIC and it is the first and most simple language to program the computers. The simulator provided with the BASCOM-AVR is also very effective and easy to use. This IDE is the easy and fastest option to start with programming the BASCOM-AVR. Few sample programs are written in BASCOM-AVR.

Microsoft Visual Studio 2008 supports latest .NET framework 3.5 and is the IDE for software development from Microsoft Corporation. The VS 2008 uses import wizard for transferring the VC++ code resulting from SDK plugin Wizard. After importing the code the file indicated in the figure 8.13 is edited or required files have to be included. But this generated DLL will be supported by .NET runtime framework.

9.4 Maintenance

After using the product regularly it is obvious that it will require maintenance. Similar is applicable to the microcontrollers, the program code that is to be flashed on the target microcontroller corrupts the Flash and EEPROM of both ATmega16 and ATmega128 after long use. The flash and EEPROM has some life of erase and write. As the target is used for debugging purpose the life of both the storage memories may expire quickly, so it is important to notice the behavior of the target microcontroller if it gives unpredictable responses.

It is also found that the device that is used in debugging purposes is not used for production purposes. Thus in real there no maintenance than this, but if prototype area is used for development then the wiring done on it should not be allowed to touch other parts of the ATX cabinet. No cooling is required here as the target chip does not heat up above normal conditions. Ultimately this section is says to change the target microcontroller after it starts malfunctioning or the coding can use flash addresses that are not used frequently, it depends if PCI card is used single handedly.



Chapter 10

Conclusion and future work.

Chapter 10 Conclusion and Future work

10.1 This section aims to critically evaluate the choices made relating to project organization issues, design decisions, project shortcomings and successes, as well as problems encountered. All in all, no serious problem was encountered during the research work. Much because the source was freely available and support from the people who engineered the system is possible through public mailing list. Moreover, one can always get help and valuable ideas from the countless people all over the world involved in designing and development of programmers and debuggers. Our proposed modification should be sufficient to support any further development of PCI card.

The project plan was useful in providing a list of project stages and tasks to be completed. The PCI chip selection and testing time have been broken down and specified as distinct time slots instead of being grouped together as the large. This however was necessary at the early stage of planning due to not knowing in detail which modules were needed, and was rectified by frequent reviews of the PCI chips to be completed in the project's development record.

The choice of microcontroller was a critical issue before beginning the research work, as it has to meet the latest requirements of debugging and JTAG boundary scan techniques which are supported by large number of processors available. The selection of 8 – bit microcontroller was decided as to develop the PCI card for educational and research use only.

The choice of higher level language was obvious for the faster development process, C language for AVR microcontroller and C# for developing GUI applications with AVR Studio SDK. Although smaller examples are taken on the assembly language of AVR microcontroller supported by the assembler of AVR Studio.

The main purpose of this thesis was to design and test a low cost PCI embedded card for microcontroller trainer, which is capable of handling

different kinds of communication protocols to serve as a learning tool for students and researchers. Initial effort went into an extensive research for finding similar PCI chips used for educational purposes. However, the result of the research was that there were very few or no boards which could serve the purpose of including all types of learning modules into one board and serve as platform for students and researchers to work on. The design and testing of this PCI card has been done in order to manufacture a simple, low cost, small in size, easy to handle and versatile microcontroller board which holds various communication modules capable of communicating with another similar board or with a PC or act as a link between two PCs. Students and researchers will find this board very useful to learn basic embedded systems and communication theories by conducting lab exercises and other experiments. Although a lot of effort and time has gone into the design of the board and then further testing of the hardware, the board is still in a stage where future work needs to be done. Major effort was made in the initial design of the board, including the selection of hardware and other design related components. The effort towards this thesis can be summarized as follows:

- Initial research for PCI cards available in the market.
- Research on various communication modules such as serial UART, infrared, AT – commands and SPI, JTAG protocols.
- Preparing lab exercise for the students to work on to ensure that all the modules are helpful as a learning tool for embedded system students.
- Initial research on available hardware for PCI bus interface.
- Selection for ideal microcontroller for the trainer system.
- Schematic design for the card.
- Writing test code and driver software for different modules.
- Testing of the hardware with different evaluation boards to check the correctness of our design
- Making design changes to the schematic based on the test results
- Creating foot prints of the components for layout design.

The PCI card design was completed and further testing was done using each module to ensure that the manufactured version of the board will be workable.

10.2 Future work

The future work in this thesis may include the manufacture of the board and performing direct tests on the communication and peripheral modules on the board. Data rates and limit to which the data rate can be pushed for normal working of the board can be done. Students taking up embedded systems and advanced embedded systems should be strongly encouraged to take this up as a project to test the working model of the PCI embedded card. Sample lab exercises have been provided in the Appendix G of this thesis. It will serve as a base for students to work on the different data communication modules and peripherals test the data rates using serial UART and I/O pins. Measurement of error rates at different speeds can be carried out.

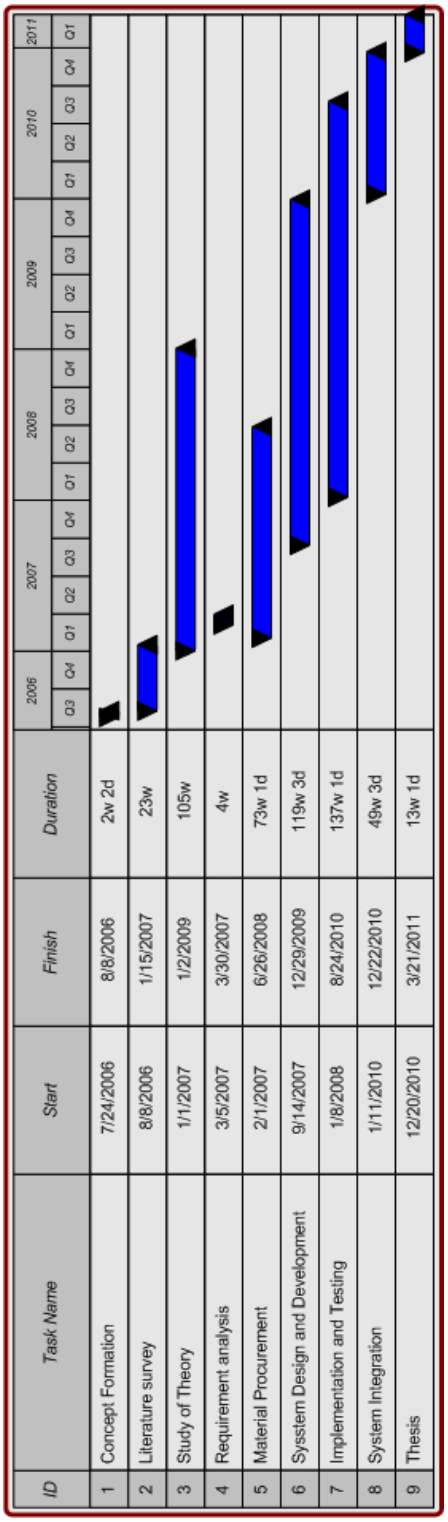
Chapter 11

Appendices

Gantt chart.
Schematics.
Bill of materials
Datasheets
Figures.
Abbreviations.
Sample exercise
References.
CD Contents
Paper published and Conferences Attended.

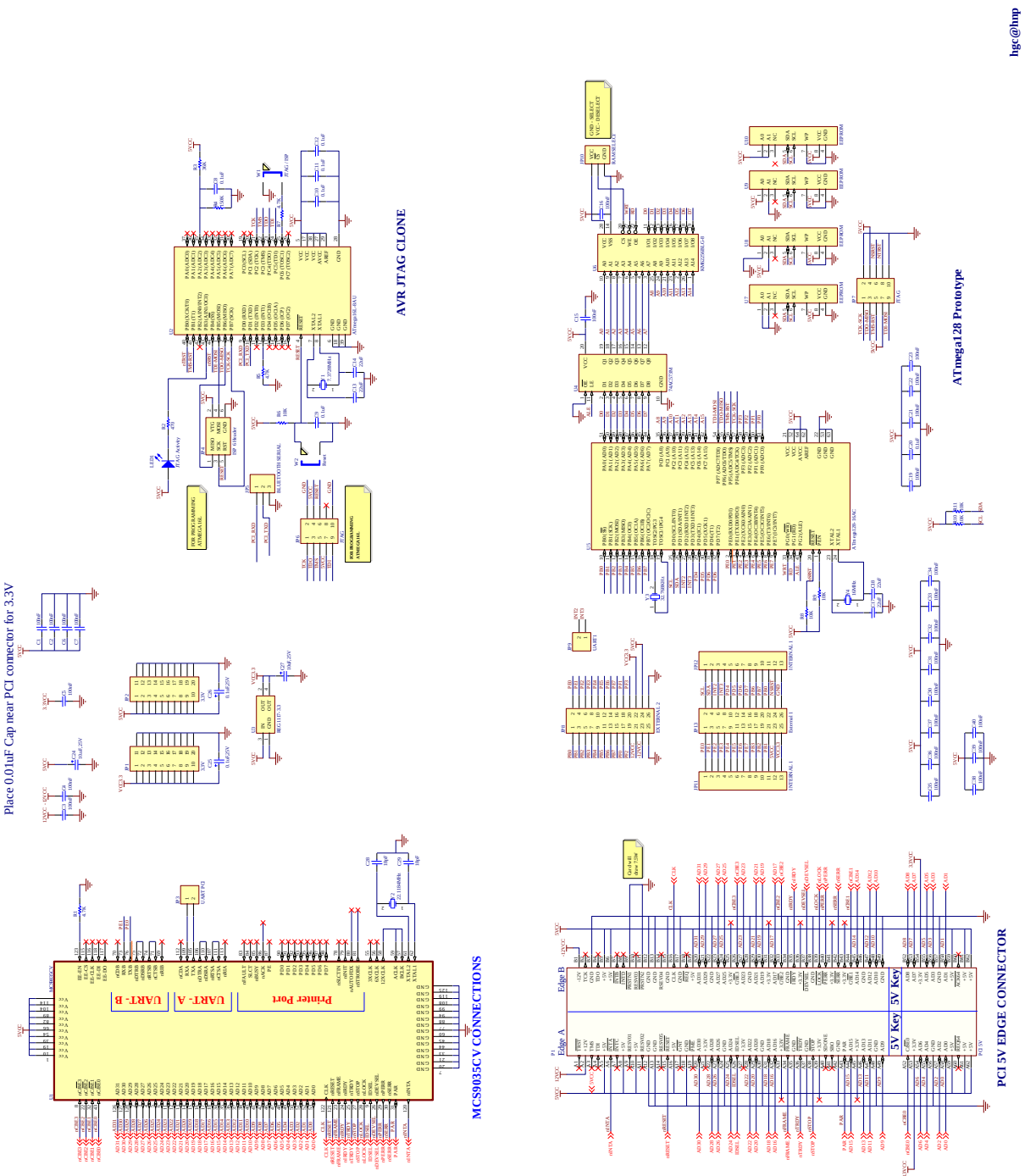
Appendix

A. Gant Chart.



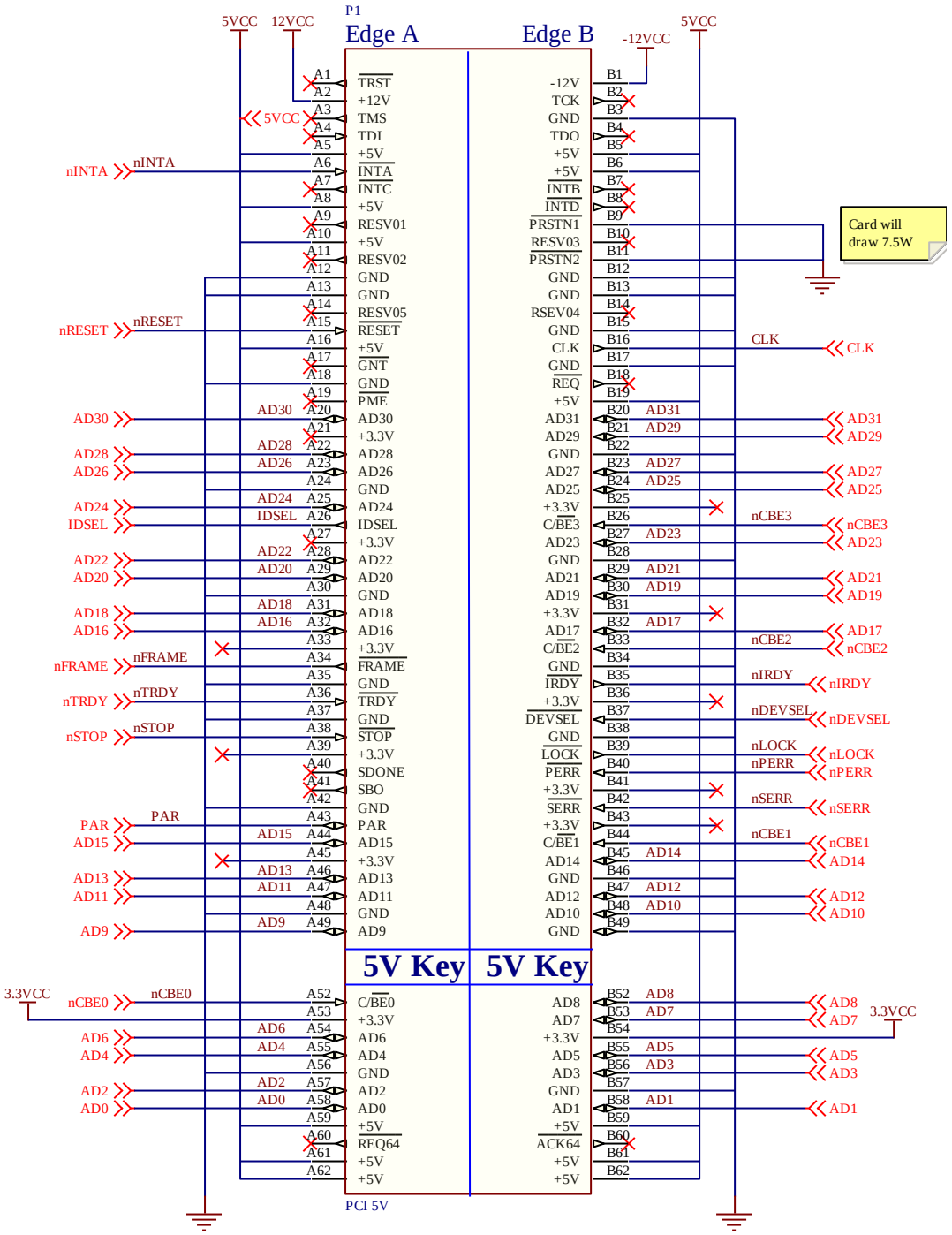
ID	Task Name	Start	Finish	Duration
1	Concept Formation	7/24/2006	8/8/2006	2w 2d
2	Literature survey	8/8/2006	1/15/2007	23w
3	Study of Theory	1/1/2007	1/2/2009	105w
4	Requirement analysis	3/5/2007	3/30/2007	4w
5	Material Procurement	2/1/2007	6/26/2008	73w 1d
6	Systtem Design and Development	9/14/2007	12/29/2009	119w 3d
7	Implementation and Testing	1/8/2008	8/24/2010	137w 1d
8	System Integration	1/11/2010	12/22/2010	49w 3d
9	Thesis	12/20/2010	3/21/2011	13w 1d

B. Schematics.



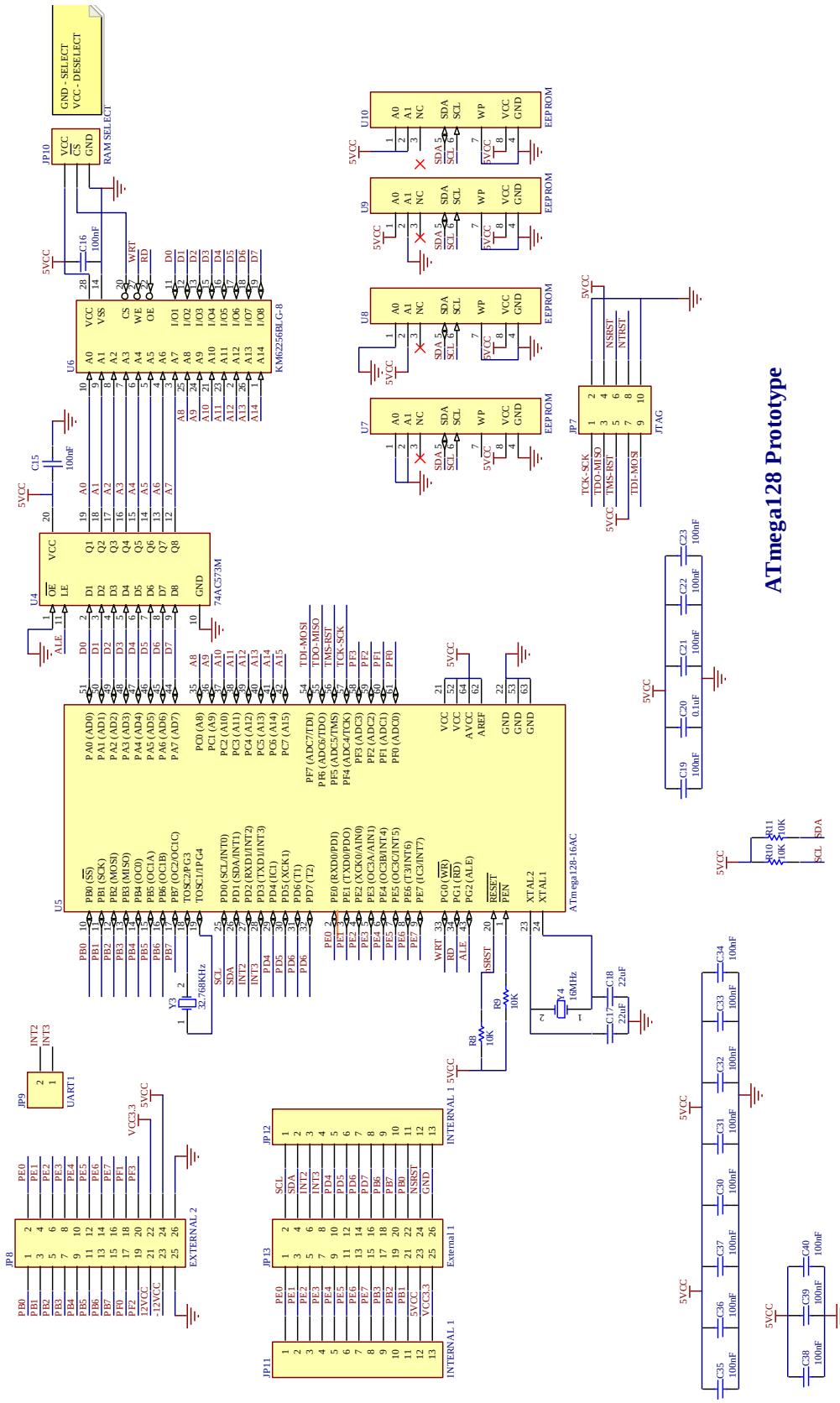
B.1 Full Schematic.

B.2 PCI Connectors

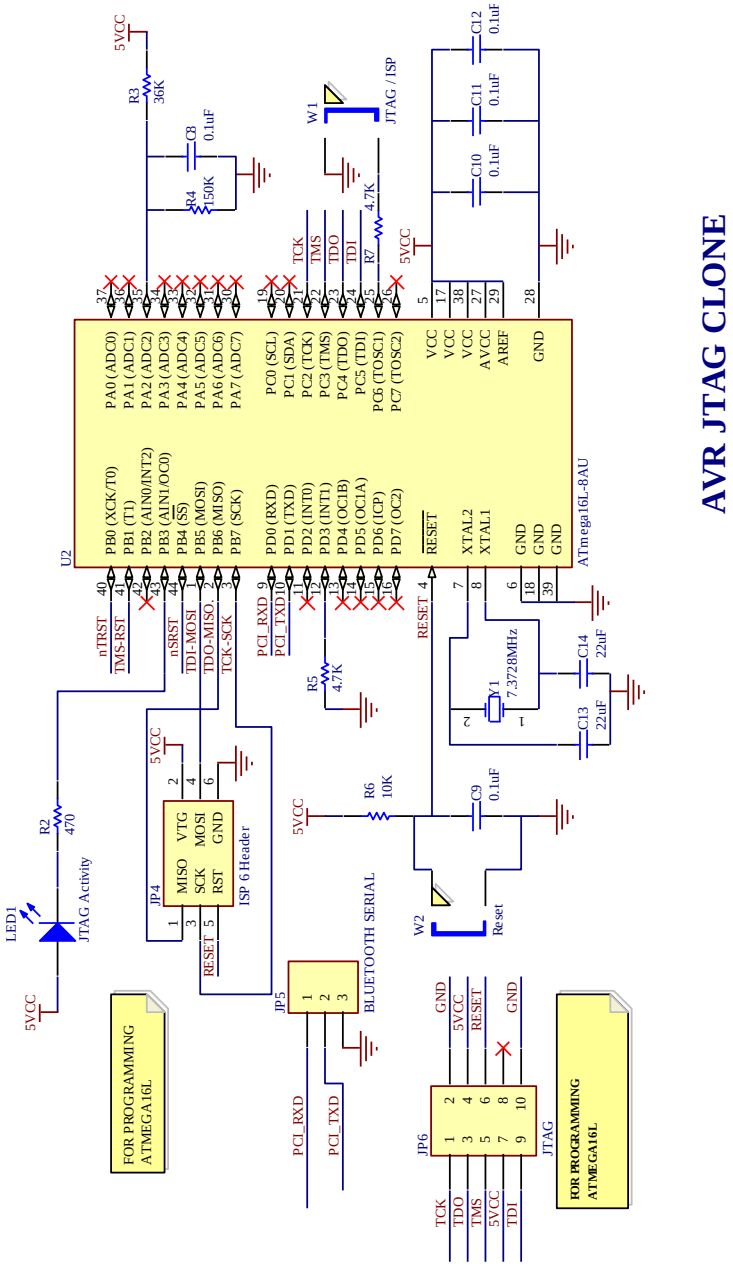


PCI 5V EDGE CONNECTOR

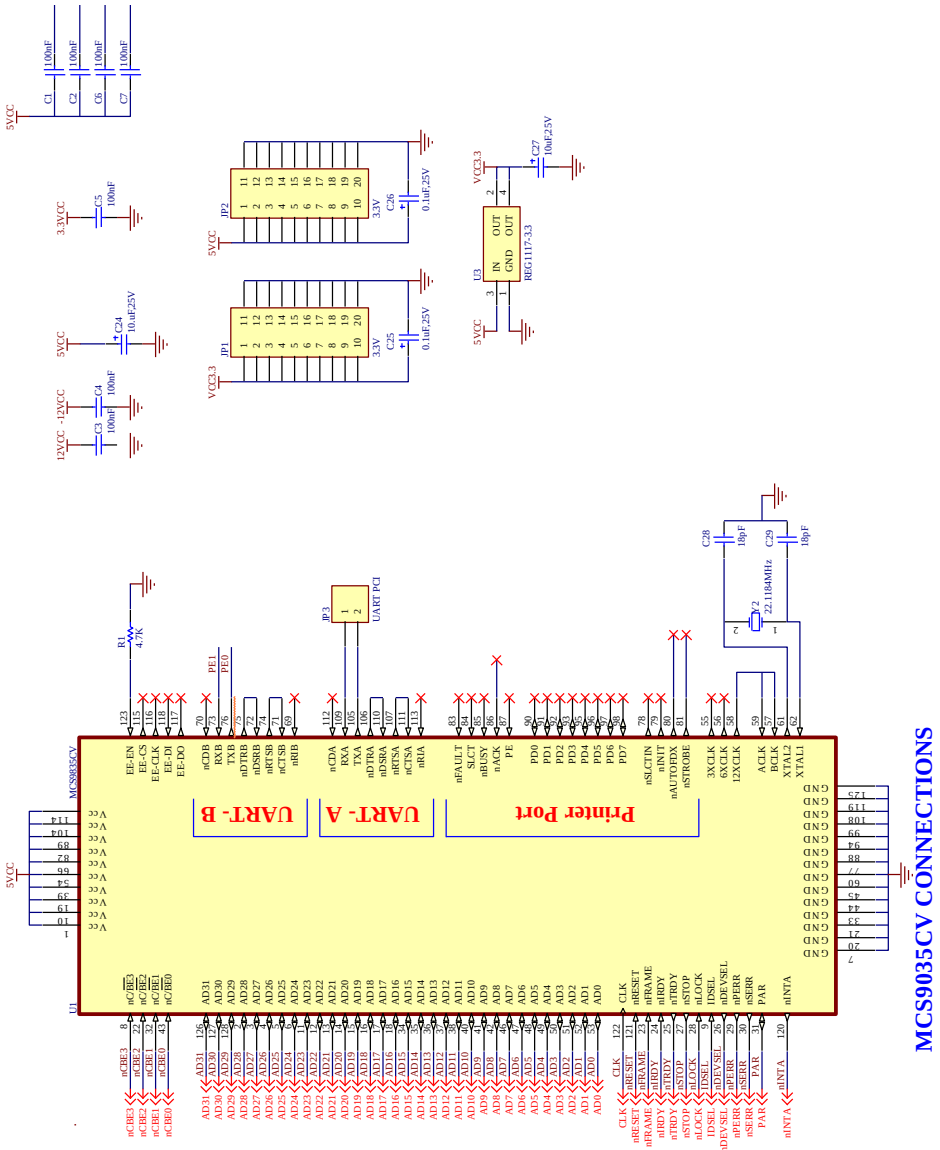
B.3 ATmega128 and Interfaces

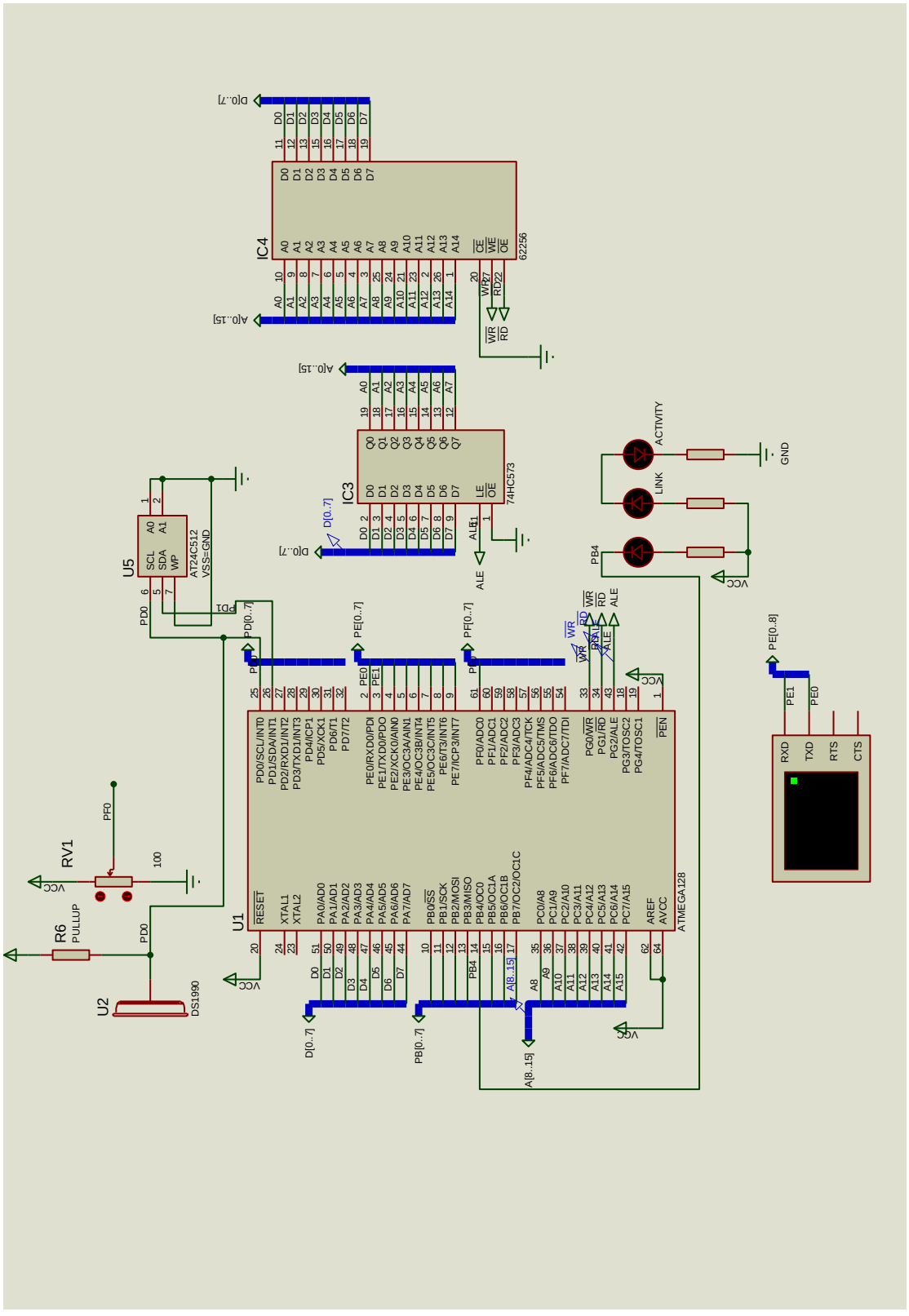


B.4 ATmega16L



B.5 MCS9835CV Interfacing





B.6 ATmega128 Interfacing modules.

C. Bill of materials (BOM)

Description	Designator	Footprint	Quantity	Value
Capacitor	C1	CC3216-1206	1	100nF
Capacitor	C2	CC3216-1206	1	100nF
Capacitor	C3	CC3216-1206	1	100nF
Capacitor	C4	CC3216-1206	1	100nF
Capacitor	C5	CC3216-1206	1	100nF
Capacitor	C6	CC3216-1206	1	100nF
Capacitor	C7	CC3216-1206	1	100nF
Capacitor	C8	CC3216-1206	1	0.1uF
Capacitor	C9	CC3216-1206	1	0.1uF
Capacitor	C10	CC3216-1206	1	0.1uF
Capacitor	C11	CC3216-1206	1	0.1uF
Capacitor	C12	CC3216-1206	1	0.1uF
Capacitor	C13	CC3216-1206	1	22uF
Capacitor	C14	CC3216-1206	1	22uF
Capacitor	C15	CC3216-1206	1	100nF
Capacitor	C16	CC3216-1206	1	100nF
Capacitor	C17	CC3216-1206	1	22uF
Capacitor	C18	CC3216-1206	1	22uF
Capacitor	C19	CC3216-1206	1	100nF
Capacitor	C20	CC3216-1206	1	0.1uF
Capacitor	C21	CC3216-1206	1	100nF
Capacitor	C22	CC3216-1206	1	100nF
Capacitor	C23	CC3216-1206	1	100nF
Polarized Capacitor (Radial)	C24	CAPPR2-5X6.8	1	10.uF,25V
Polarized Capacitor (Radial)	C25	CAPPR2-5X6.8	1	0.1uF,25V
Polarized Capacitor (Radial)	C26	CAPPR2-5X6.8	1	0.1uF,25V
Polarized Capacitor (Radial)	C27	CAPPR2-5X6.8	1	10uF,25V

Capacitor (Semiconductor SIM Model)	C28	CC3216-1206	1	18pF
Capacitor (Semiconductor SIM Model)	C29	CC3216-1206	1	18pF
Capacitor	C30	CC3216-1206	1	100nF
Capacitor	C31	CC3216-1206	1	100nF
Capacitor	C32	CC3216-1206	1	100nF
Capacitor	C33	CC3216-1206	1	100nF
Capacitor	C34	CC3216-1206	1	100nF
Capacitor	C35	CC3216-1206	1	100nF
Capacitor	C36	CC3216-1206	1	100nF
Capacitor	C37	CC3216-1206	1	100nF
Capacitor	C38	CC3216-1206	1	100nF
Capacitor	C39	CC3216-1206	1	100nF
Capacitor	C40	CC3216-1206	1	100nF
Header, 10-Pin, Dual row	JP1	HDR2X10_CEN	1	
Header, 10-Pin, Dual row	JP2	HDR2X10_CEN	1	
Header, 2-Pin	JP3	HDR1X2	1	
Header, 3-Pin, Dual row	JP4	HDR2X3	1	
Header, 3-Pin, Right Angle	JP5	HDR1X3	1	
Header, 5-Pin, Dual row	JP6	HDR2X5	1	
Header, 5-Pin, Dual row	JP7	HDR2X5	1	
Header, 13-Pin, Dual row	JP8	HDR2X13	1	
Header, 2-Pin	JP9	HDR1X2	1	

Header, 3-Pin	JP10	HDR1X3	1	
Header, 13-Pin	JP11	HDR1X13	1	
Header, 13-Pin	JP12	HDR1X13	1	
Header, 13-Pin, Dual row	JP13	HDR2X13	1	
Typical LED	LED1	CD3216-1206	1	
5V PCI CONNECTOR	P1	PCI5V32BIT	1	
Resistor	R1	CR3216-1206	1	4.7K
Resistor	R2	CR3216-1206	1	470
Resistor	R3	CR3216-1206	1	36K
Resistor	R4	CR3216-1206	1	150K
Resistor	R5	CR3216-1206	1	4.7K
Resistor	R6	CR3216-1206	1	10K
Resistor	R7	CR3216-1206	1	4.7K
Resistor	R8	CR3216-1206	1	10K
Resistor	R9	CR3216-1206	1	10K
Resistor	R10	CR3216-1206	1	10K
Resistor	R11	CR3216-1206	1	10K
PCI 2 UART + 1 LPT	U1	FR-QFP14x20-G128/X.3N	1	MCS9835CV
8-Bit AVR Microcontroller with 16K Bytes Flash	U2	44A	1	ATmega16L
800mA and 1A Low Dropout Positive Regulator 2.85V, 3.3V, 5V, and Adjustable	U3	ZZ311	1	LM1117-3.3
Octal D-Type Latch with 3-State Output Non-	U4	SO20B	1	74AC573

Inverting				
8-Bit AVR Microcontroller with 128K Bytes of In-System Programmable Flash Memory	U5	64A	1	ATmega128
32KBYTE SRAM	U6	KM62256	1	62256
32KBYTE EEPROM	U7	8S1	1	AT24C512
32KBYTE EEPROM	U8	8S1	1	AT24C512
32KBYTE EEPROM	U9	8S1	1	AT24C512
32KBYTE EEPROM	U10	8S1	1	AT24C512
Jumper Wire	W1	HDR1X2	1	
Jumper Wire	W2	HDR1X2	1	
Crystal Oscillator	Y1	BCY-W2/E4.7	1	7.3728MHz
Crystal Oscillator	Y2	BCY-W2/E4.7	1	22.1184MHz
Crystal Oscillator	Y3	BCY-W2/D3.1	1	32.768KHz
Crystal Oscillator	Y4	BCY-W2/E4.7	1	16MHz

D. Datasheets

The datasheets of the following ICs is available in the CD – ROM.

ATmega16L

ATmega128

62256

AT24C512

MCS9835CV

LM1117-3.3

74AC573

E. List of Figures.

F. Abbreviations.

ASIC	Application Specific IC. An IC manufactured for a company based on a unique design given by the company, but done in a standard chip process.
AGP	Accelerated Graphics Port. Standard that specifies a high bandwidth connection between the graphics controller and the main memory.
CardBus	Laptop version of the PCI bus.
CompactPCI	Passive backplane bus that attempts to replace VME. Cards have the same size as VME cards.
CMOS	Complementary MOS. A chip manufacturing technology widely used by almost all chip manufacturers today.
CPLD	Complex PLDs. A programmable logic chip based on linking multiple PLD blocks on the same chip.
DMA bus	Direct Memory Access. A method for transferring data from one agent to another without CPU intervention.
EISA	Enhanced ISA. An industry standard invented by Compaq to replace the aging ISA standard.
FPGA	Field Programmable gate Arrays. A programmable logic chip based on a matrix of basic logic cells.
GPIB	A standard interface for test and measurement equipment. Invended by HP, and adopted by IEEE as IEEE 488.
HDL	Hardware Description Language. A generic name for a computer language used to describe digital circuits for CPLD, FPGA and ASIC design.
HiRel	PCI Standard currently developed by IEEE. It supports SCI.
IDE	Integrated Drive Electronics. A mass storage interface standard, roughly based on integrating the original IBM PC hard disk controller into the drive itself.
ISA	Industry Standard Architecture. The original bus designed for the IBM PC and IBM AT.
JTAG	Joint Test Action Group. A group dedicated to setting industry standards related to electronic testing. Also a name of a computer interface for in-system testing of chips.

MESI	Modified/Exclusive/Shared/invalid. A bus coherency protocol used to guarantee cache coherency in multiprocessing systems.
MOS	Metal Oxide Semiconductor. A type of transistor used on almost all the chips manufactured today.
NuBUS	A computer bus widely in use by the Macintosh II series of computers.
PAL	Programmable Array Logic. A synonym for PLD.
PC/104	The embedded system version of the ISA bus. Also the name of the consortium that serves as a custodian of the PC/104 standard.
PC/104-Plus	Standard that specifies PC/104 size cards with both ISA and PCI bus.
PC Card	New name for the 1994 release of the PCMCIA standard.
PCI	Peripheral Component Interconnect. A local bus standard.
PCI-SIG	PCI Special Interest Group. An association of members of the microcomputer industry established to monitor and enhance the development of the PCI bus.
PCI-ISA	A passive backplane standard based on a full length CPU card containing both an ISA and a PCI connector.
PCMCIA	Personal Computer Memory Card International Association. Organization that developed the PCMCIA standard, a laptop oriented expansion bus.
PICMG	PCI Industrial Computer Manufacturers Group. A consortium of computer product vendors established to extend the PCI specification for use in industrial computing applications. Developed the PCI-ISA Passive Backplane and CompactPCI standards.
PISA	A passive backplane standard based on a half length CPU card containing an EISA like connector with ISA signals on the top row and PCI signals on the bottom row.
PLD	Programmable Logic Device. A logic chip based on a programmable AND-OR array linking a number of input and output pins.

PLI	Procedural Language Interface. A standard API for linking C modules with Verilog programs.
PMC	PCI Mezzanine Card. Defined as IEEE standard P1386.1, PMC cards use PCI chips and may be mounted on VME cards.
PXI	PCI eXtentions for Instrumentation. A VXI like bus based on CompactPCI.
RTL	Register Transfer Level. A logical abstraction level of a digital integrated circuit description, which can be easily translated to a real circuit.
SCI	Scalable Coherent Interface. IEEE standard 1596-1992. Specifies a method of interconnecting multiple processing nodes.
SCSI	Small Computer System Interface. A popular standard for linking several mass storage devices to a computer.
SmallPCI CardBus.	Expansion card with form factor identical to PC Card and Primarily intended for OEM products.
VESA	Video Electronics Standards Association. A technical forum setting PC computer graphics related standards.
VHDL	VHSIC Hardware Description Language. A popular HDL inspired by Ada, and designed to be a Military standard. It is now IEEE-1076
Verilog	A popular HDL inspired by C, and designed by Gateway corporation (now owned by Cadence). It is now IEEE-1364.
VGA	Video Gate Array. A graphics card designed by IBM, later adopted as a baseline standard. All modern graphic cards have a basic VGA compatible mode.
VITA	VME International Trade Association. The organization of VME manufacturers.
VME	Versa Module Eurocard. Passive backplane bus.
VXI	VME eXtension for Instrumentation. A test and measurement bus based on 9U VME cards.
X86	A generic name for the Intel 16/32 bit architecture implemented by the 8088 through Pentium II series of micro processors. X86

	compatible microprocessors are also implemented by other companies such as IDT, Cyrix, IBM, AMD and SGSThomson.
IEEE	Institute of Electrical & Electronics Engineers
IR	Instruction Register
ISP	In-System Programming
JTAG	Joint Test Action Group
MCM	Multi-Chip Module
Mfg	Manufacturing
PCB	Printed Circuit Board
PRPG	Pseudo-Random Pattern Generation
PSA	Parallel Signature Analysis
PWB	Printed Wiring Board
SPL	Scan Path Linker
SVF	Serial Vector Format
TAP	Test Access Port
TBC	Test Bus Controller
TCK	Test Clock
TDI	Test Data Input
TDO	Test Data Output
TMS	Test Mode Select
TRST	Test Reset
UUT	Unit Under Test
ASP	Addressable Scan Port
ATE	Automatic Test Equipment
ATPG	Automatic Test Pattern Generation
BIST	Built-In Self-Test
B/S	Boundary-Scan
BSC	Boundary-Scan Cell
BSDL	Boundary-Scan Description Language
BSR	Boundary-Scan Register
BST	Boundary-Scan Test
CAE	Computer-Aided Engineering
DFT	Design-for-Test
DR	Data Register

DSP	Digital Signal Processing/Processor
EDA	Electronic Design Automation
eTBC	Embedded Test Bus Controller
FPGA	Field-Programmable Gate Array
HSDL	Hierarchical Scan Description Language
ICE	In-Circuit Emulation
ICT	In-Circuit Test

G. Sample exercise

The sample exercises given here are targeted to ATmega16L microcontroller, the VCC and GND are assumed to be at +5V V_{cc}. Few examples pertaining to AVR programming are mentioned below with circuit diagram and program listing in C language, AVR assembly and BASIC.

G.1 Flashing LEDs.

The circuit in figure E.1 shows the LED connected to the PORTD of the ATmega16L microcontroller. In this circuit the LEDs glow sequentially from left to right and right to left. The comments in the program explains the working.

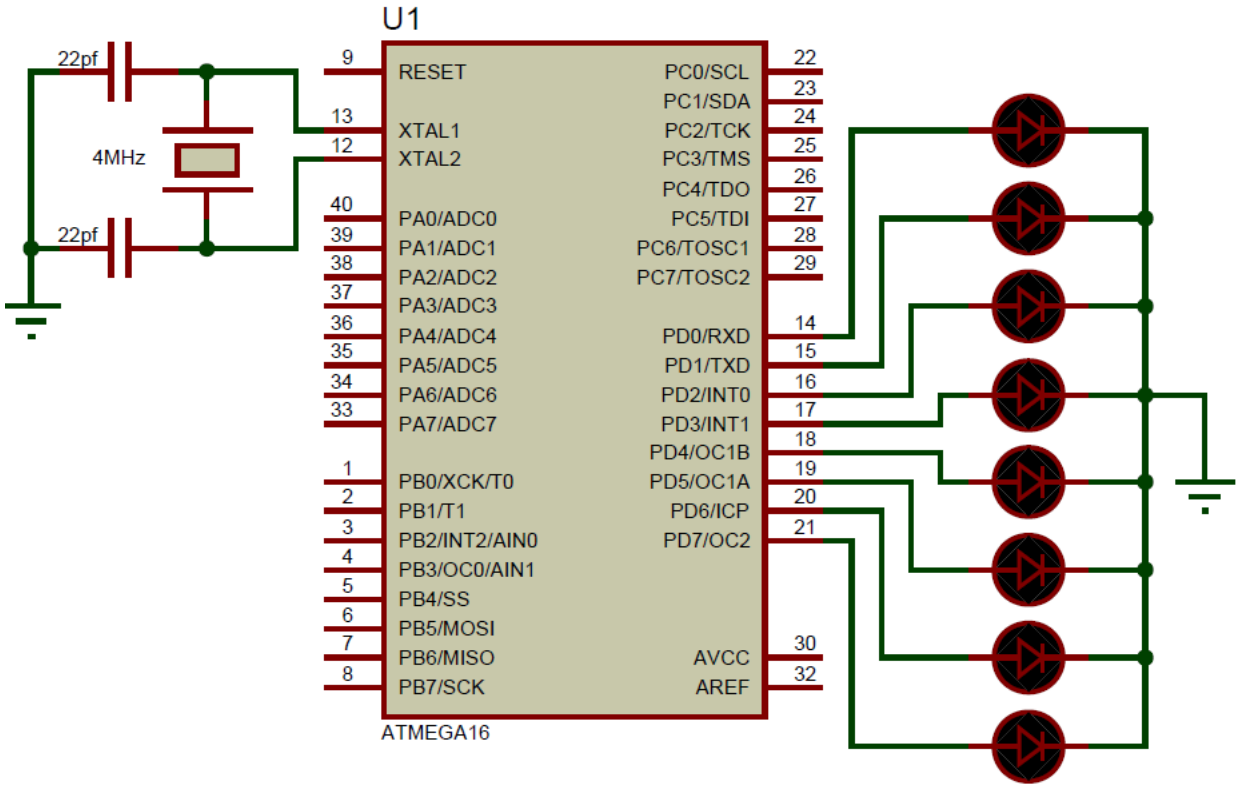


Figure G.3 Flashing LED

The program listing is given below:

```
int i;
void main (void)
{
      DDRD = 0xFF;//PORT D configured output
      while(1)
      {
            for(i = 1; i <= 128; i = i*2)
            {
```

```
                PORTD = i;
                Delay(100);
            }
            for(i = 128; i > 1; i = i/2)
            {
                PORTD = i;
                Delay(100);
            }
      }
}
void Delay(unsigned long millisec)//approx. delay
{
    int i;
    while (millisec--)
      {
        for (i=0; i<125; i++)
                asm volatile ("nop"::);
      }
```

G.2 Seven segment with UP counter

The circuit in figure E.2 demonstrates hexadecimal Up counter using common cathode 7-segment digit display. The counter is incremented by a push to on switch at pin PORTC0.

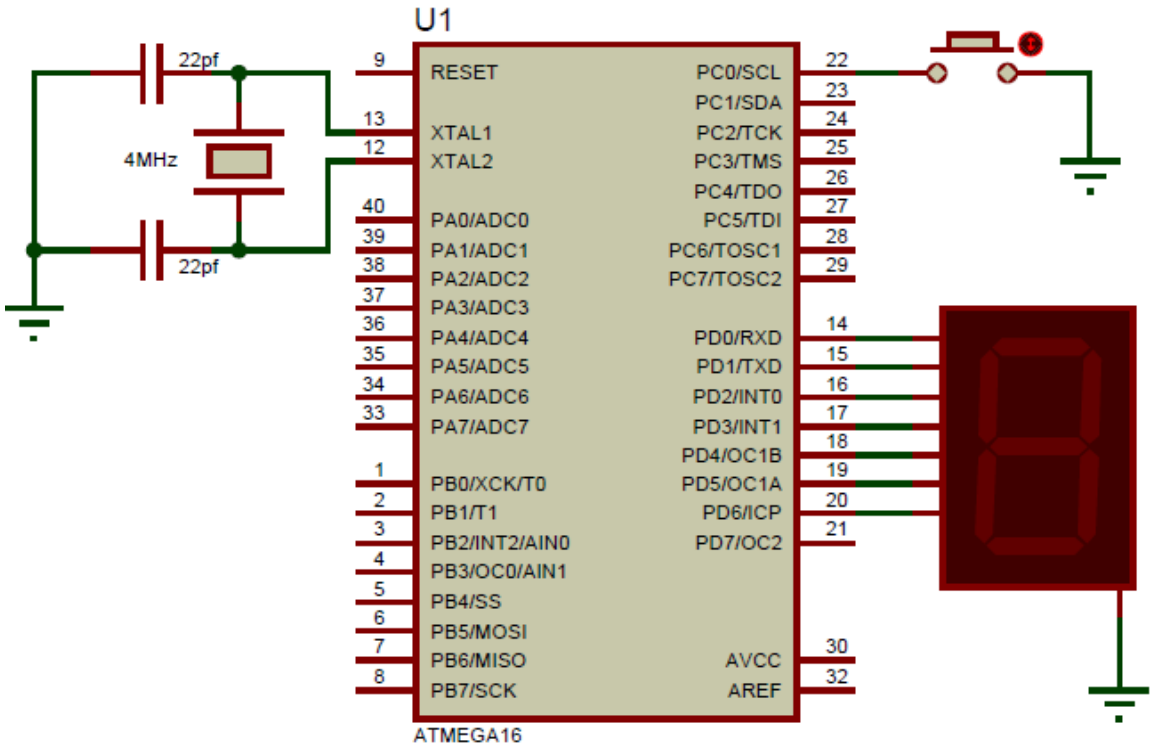


Figure G.2 Seven Segment with UP counter.

The program listing for above circuit is given below:

```
char digits[16]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,
0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71};
unsigned char;
unsigned char p_state;
unsigned char key;
unsigned char i;
void main(void)
{
      DDRD = 0xFF;
      PORTD = digits[0];
      DDRC = 0x00;
      PORTC = 0xFF;
      while(1)
      {
            if(!PINC.0)
            {
```

```
                    if(key!=p_state)
                    {
                            if(i==15)
                            {
                                    i=0;
                                  PORTC=digits[0];
                          }
                          else
                          i++;
                            PORTD = digits[i];
                          p_state=key;
                    };
            }
            else
              p_state=0xFF;
      }
}
```

G.3 Multiplexed Keypad

Here in circuit figure G.3 the variable key has the actual number of the keypad scanned.

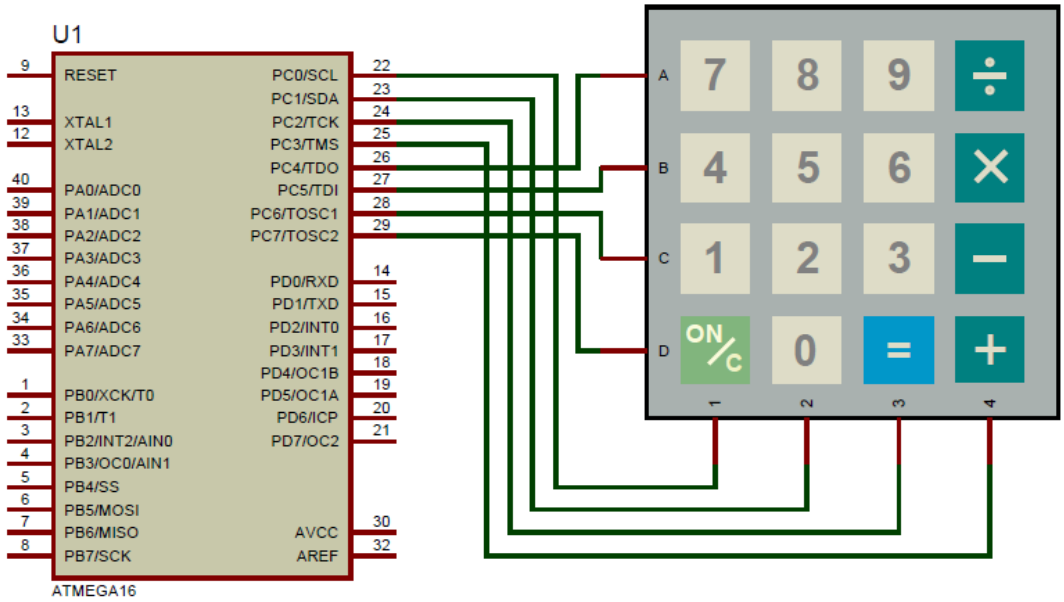


Figure G.3 Multiplexed Keyboard

The program listing is given below:

```
#include <mega16.h>
#include <delay.h>

#define maxkeys 16

unsigned char key, butnum;

//key pad scan table
flash unsigned char keytbl[16]={0xee, 0xed, 0xeb, 0xe7, 0xde, 0xdd,
0xdb, 0xd7, 0xbe, 0xbd, 0xbb, 0xb7, 0x7e, 0x7d, 0x7b, 0x77};

void main(void)
  {
  //Init port B to show keyboard result
  DDRB = 0xff;
  PORTB = 0xff;

  //endless loop to read keyboard
  while(1)
      {
```

```
    //get lower nibble
     DDRC = 0x0f;
     PORTC = 0xf0;
     delay_us(5);
     key = PINC;

     //get upper nibble
     DDRC = 0xf0;
     PORTC = 0x0f;
     delay_us(5);
     key = key | PINC;

     //find matching keycode in keytbl
     if (key != 0xff)
     {
        for (butnum=0; butnum<maxkeys; butnum++)
        {
            if (keytbl[butnum]==key)  break;
        }
        if (butnum==maxkeys) butnum=0;
          else butnum++; //to make range 1-16
     }
     else butnum=0;

     PORTB = ~ butnum ;

    } // end while
}   //end main
```

G.4 16 x 2 LCD programming

The circuit figure G.4 shows the 16 x 2 LCD programming.

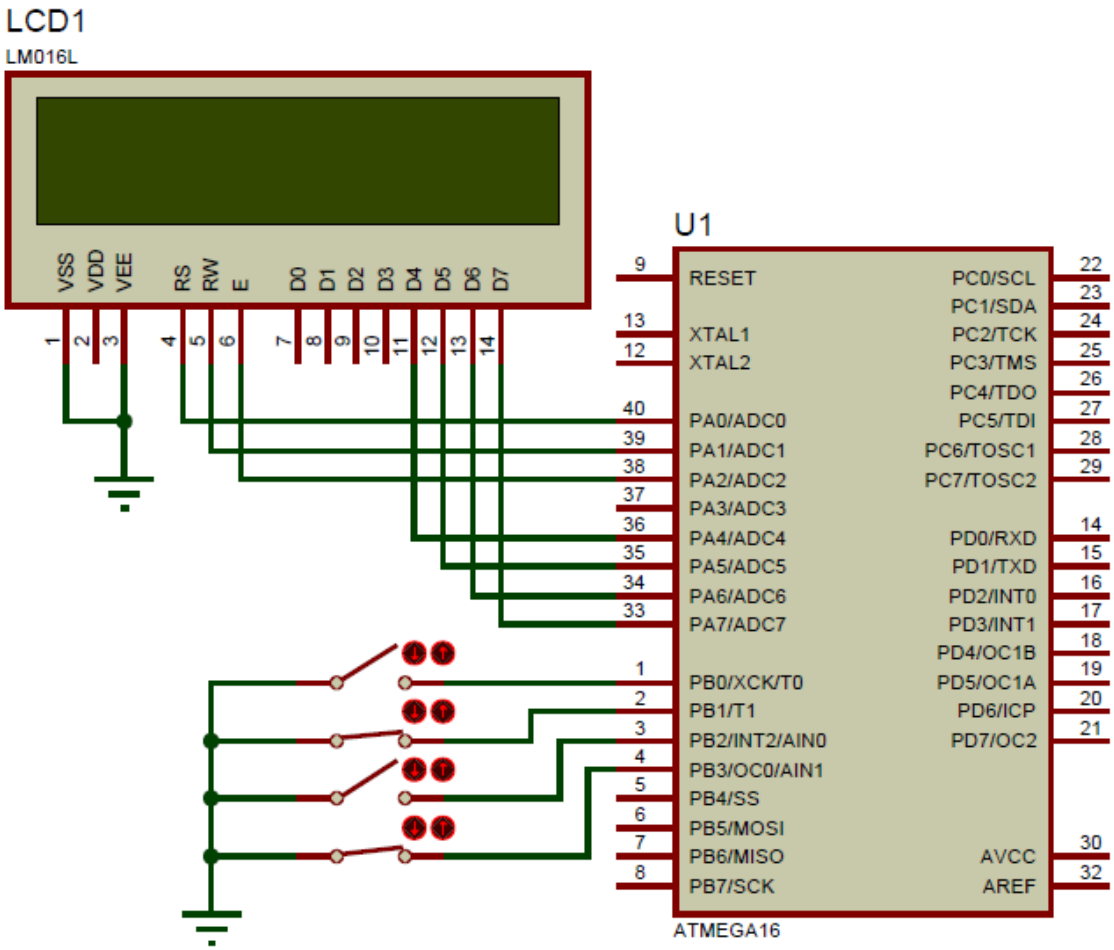


Figure G.4 16 x 2 LCD programming.

The demonstration program is listed below:

```
#include <mega16.h>
#include <delay.h>
#include <stdio.h>
#include <lcd.h>

#asm
   .equ __lcd_port=0x1B ;PORTA
#endasm

void main(void)
{
    char buffer[20];
    unsigned char w;
```

```
    PORTB=0xFF;
    DDRB=0x00;

    lcd_init(16);
    lcd_clear();
    lcd_putsf("Set Keys");

    while (1){

        w = ~PINB;

        if(w!=0x00)
        {
            lcd_clear();
            lcd_gotoxy(0,0);
            sprintf(buffer,"Number=%d",w);
            lcd_puts(buffer);
            delay_ms(100);
         }
         else
         {
            lcd_clear();
            lcd_putsf("Number=0");
            delay_ms(100);
         }
    }
}
```

G.5 PWM LED dimming example

The figure B.6 shows the LED connected to PB4 when using Ext2 header with External interfacing board of figure 8.12. The listing is shown below.

```
;THIS PROGRAM SETS THE DIMMING OF LED USING SERIAL PORT
.INCLUDE "M128DEF.INC"
.CSEG
.SET LED=PB4
.DEF SE_DT_IN=R23
.DEF SE_DT_OUT=R22
.ORG 0X00
      JMP RESET
.ORG 0X024
      JMP USART_RXC
.ORG 0X02A
RESET:      LDI R16,$FF ;SET THE STACK POINTER TO 0X10FF
            OUT SPL,R16
            LDI R16,$10
            OUT SPH,R16
            CALL INIT_UART    ;INITIALIZE THE UART
            CALL INIT_PWM     ;INITIALIZE THE PWM

            SEI               ;ENABLE GLOBAL INT
RE:         NOP
            ;LDI SE_DT_OUT,0X41
            ;CALL TXD_TO_PC
            RJMP RE
.ORG 0X100
;++++++++++++++++++++++++++++++++++++++++++++++++++++
;ROUTINE TO INITIALIZE UART
INIT_UART:  CLR R17
                  STS UBRR0H,R17
                  LDI R18,$33       ;SET 9600 BAUD
                  OUT UBRR0L,R18
                  LDI R17,(1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0) ;ENABLE
THE
                  OUT UCSR0B,R17                    ;RECV  &  TRANS
RNABLE BITS
                  LDI
R18,(0<<UMSEL0)|(0<<UPM00)|(0<<USBS0)|(3<<UCSZ00);8BIT,NONE  PARITY  1
STOPBIT
```

```
                STS UCSR0C,R18

                RET
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;ROUTINE TO RECEIVE THE DATA AND DISPLAY
;RXD_FROM_PC:    SBIS UCSRA,RXC
;                      RJMP RXD_FROM_PC
;                      IN SE_DT_IN,UDR
;                      RET
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;ROUTINE TO TRANSMIT DATA TO PC
TXD_TO_PC:
                SBIS UCSR0A,UDRE0
                RJMP TXD_TO_PC
                OUT UDR0,SE_DT_OUT
                nop
                SBI UCSR0A,UDRE0
                nop
                RET
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++;
;ROUTINE FOR RXD INTERRUPT COMPLETE
USART_RXC:  IN SE_DT_IN,UDR0
                OUT OCR0,SE_DT_IN ;SETTING THE CONTRAST DIRECTYLY
                MOV SE_DT_OUT,SE_DT_IN;ECHO TO THE PC
                CALL TXD_TO_PC
                LDI    R18,(1<<RXCIE0)|(1<<RXEN0)|(1<<TXEN0);ENABLE
RXD INT
                OUT UCSR0B,R18
                RETI
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;ROUTINE FOR PWM     PHASE CORRECT 8-BIT
INIT_PWM:   LDI R16,(1<<WGM00)|(1<<COM01)|(1<<CS02)
                OUT TCCR0,R16
                LDI R16,$20
                OUT OCR0,R16      ;SET DUTY CYCLE
                SBI DDRB,LED
                RET
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
DELAY:      LDI XH,$5
BIG:  SER XL
SMALL:      NOP
```

```
            DEC   XL
            BRNE SMALL
            DEC XH
            BRNE BIG
            RET
.EXIT
```

G.6 Sampling ADC0 using UART0

Here the program listing in BASCOM-AVR is written for sampling ADC0 at pin PF0 as shown in figure B.6.

```
'-----------------------------------------------------------------------
'                   ADC_INT.BAS
'   demonstration of GETADC() function in combintion with the idle
mode
'  for a better noise immunity
'   Getadc() will also work for other AVR chips that have an ADC
converter
'-----------------------------------------------------------------------
$regfile = "m128def.dat"
$crystal = 8000000
$hwstack = 40
$swstack = 8
$framesize = 40
$baud = 9600
Config Com1 = 9600 , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
'configure single mode and auto prescaler setting
'The single mode must be used with the GETADC() function
'The prescaler divides the internal clock by 2,4,8,16,32,64 or 128
'Because the ADC needs a clock from 50-200 KHz
'The AUTO feature, will select the highest clockrate possible
Config Adc = Free , Prescaler = Auto , Reference = Avcc
'Now give power to the chip
On Adc Adc_isr Nosave
Enable Adc
Enable Interrupts



Dim W As Word , Channel As Byte , Dd As Word , Aa As Single
   Dim Str1 As String * 6
Channel = 0
'now read A/D value from channel 0
Do
  Channel = 0
  'idle will put the micro into sleep.
  'an interrupt will wake the micro.
```

```
  Start Adc
  Idle
  Stop Adc
  Dd = W * 5
  Aa = Dd \ 1024.0
 Print "Channel " ; Channel ; " value " ; Str(aa , 2)
Loop
End
Adc_isr:
  push r26
  push r27
  push r24
  in r24,sreg
  push r24
  push r25
  W = Getadc(channel)
  pop r25
  pop r24
  !out sreg,r24
  pop r24
  pop r27
  pop r26
Return
```

G.7 Example of 1-wire iButton.

The sample for 1-wire device as shown in figure B.6 pin PD0 can also be done on the external card of figure 8.12. the program listing is done in BASCOM-AVR is given here.

```
$regfile = "m128def.dat"
$crystal = 8000000
$hwstack = 32           ' default use 32 for the hardware stack
$swstack = 10           ' default use 10 for the SW stack
$framesize = 40         ' default use 40 for the frame space
Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
'when only bytes are used, use the following lib for smaller code
$lib "mcsbyte.lib"
Config 1wire = Portd.0      'use this pin
'On the STK200 jumper B.0 must be inserted
Dim Ar(8) As Byte , A As Byte , I As Byte
Do
Wait 1
1wreset               'reset the device
Print Err            'print error 1 if error
1wwrite &H33           'read ROM command
For I = 1 To 8
   Ar(i) = 1wread()  'place into array
Next
'You could also read 8 bytes a time by unremarking the next line
'and by deleting the for next above
'Ar(1) = 1wread(8)   'read 8 bytes
For I = 1 To 8
    Print Hex(ar(i));   'print output
Next
Print                  'linefeed
Loop                   'read 8 bytes
End
```

G.8 I2C interface EEPROM

The AT24C512 EEPROM is interfaced to the I2C bus of ATmega128 as shown in figure B.6 and 7.18. the program listing is written in BASCOM-AVR given below.

```
$regfile = "m128def.dat"
$lib "i2c_twi.lbx"
$crystal = 8000000
$baud = 9600
Config Com1 = 9600 , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Config Scl = Portd.0
Config Sda = Portd.1
'connect A0  to ground, and A1 to VCC
Declare Sub Writebyte512(address As Word , Value As Byte)
Declare Function Readbyte512(address As Word) As Byte
Dim B As Byte , Adrr As Word
Dim Hdr As Byte , L As Byte , H As Byte
Config Twi = 400000                                    'I2C SPEED
I2cinit
B = 65
Print "AT24C512 EEPROM DEMO"
For Adrr = 11 To 20
  B = Adrr
  Writebyte512 Adrr , B                        'write value
Next
Waitms 100
For Adrr = 11 To 20                          'read values back
    B = Readbyte512(Adrr)
    Print B                                  'and show them
Next
End
'write a byte to the 24LC512 memory
Sub Writebyte512(address As Word , Value As Byte)
    Hdr = &B10100000
    Hdr.3 = Address.15                  ' A0,A1 Gnd
    H = High(address)
    L = Low(address)
    I2cstart
    I2cwbyte Hdr                'address with A15 segment bit
    I2cwbyte H                                             'MSB
```

```
    I2cwbyte L                                                  'LSB
    I2cwbyte Value                                              'value
    I2cstop
    Waitms 20
End Sub
'read a byte from the 24LC512 memory
Function Readbyte512(address As Word) As Byte
    Hdr = &B10100000
    Hdr.3 = Address.15                      ' A0,A1 gnd
    H = High(address)
    L = Low(address)
    I2cstart
    I2cwbyte Hdr
    I2cwbyte H
    I2cwbyte L
    I2crepstart                         'repeated start
    Hdr.0 = 1                         ' we will read now
    I2cwbyte Hdr
    I2crbyte Readbyte512 , Nack   'read byte
    I2cstop
End Function
```

G.9 XMEM parallel interface with SRAM

This repeat from section 7.5.4 of chapter 7.

```
;This example shows the access to the external memory
;along with internal memory, Here PC7 is released
.include "m128def.inc"
.CSEG
RJMP RESET
.ORG 0X46
RESET:      LDI R16,LOW(RAMEND)           ;SET THE STACK POINTER
            LDI R17,HIGH(RAMEND)
            OUT SPL,R16
            OUT SPH,R17
            LDI  R16,(1<<SRE); enable the Xmen interface
            OUT MCUCR,R16
            LDI R16,(1<<XMM0); PC7 is released due to 32K SRAM
            STS XMCRB, R16
            LDI R16, $A3
            STS $204, R16;accessing memory 0x204 of internal SRAM
            LDS  R18,$204
            com r16
            STS $8204, R16;accessing memory 0x204 of external SRAM
            LDS  R18,$8204
            inc r16
            STS $7204, R16;accessing memory 0x7204 of external SRAM
            LDS  R18,$7204
            RJMP RESET
.EXIT
```

G.10 1-Wire iButton interface

The iButton is connected to pin 25 PD0 of ATmega128 as shown in figure B.6. the program listing is written in BASCOM-AVR is give here.

```
$regfile = "m128def.dat"
$crystal = 8000000
$hwstack = 32          ' default use 32 for the hardware stack
$swstack = 10          ' default use 10 for the SW stack
$framesize = 40        ' default use 40 for the frame space
Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
'when only bytes are used, use the following lib for smaller code
$lib "mcsbyte.lib"
Config 1wire = Portd.0                 'use this pin
'On the STK200 jumper B.0 must be inserted
Dim Ar(8) As Byte , A As Byte , I As Byte
Do
Wait 1
1wreset                  'reset the device
Print Err               'print error 1 if error
1wwrite &H33             'read ROM command
For I = 1 To 8
   Ar(i) = 1wread()        'place into array
Next
'You could also read 8 bytes a time by unremarking the next line
'and by deleting the for next above
'Ar(1) = 1wread(8)       'read 8 bytes
For I = 1 To 8
    Print Hex(ar(i));       'print output
Next
Print                        'linefeed
Loop                       'read 8 bytes
End
```

The module programs are included in the CD-ROM.

Example manufactures and PCI Plug and Play ID

Manufacturer	Man. ID	PNP ID	Manufacturer	Man. ID	PNP ID
3COM	10b7	4279	SMC	10b8	4280
Acer	1025	4133	Dell	1028	4136
Acer	10b9	4281	Mitsubishi	10ba	4282
Adaptec	9004	36868			
AMD	1022	4130	Trident	1023	4131
American Mega	101e	4126	PictureTel	101f	4127
AMP	1038	4152	Seiko Epson	103a	4154
AST	100d	4109	Weitek	100e	4110
Award	10c4	4292	Xerox	10c5	4293
Compaq	1032	4146	NEC	1033	4147
Creative Labs	1102	4354	Santa Cruz	1111	4369
Cyrix	1078	4216	I-BUS	1079	4217
Daewood	1070	4208	Mitac	1071	4209
Data General	1089	4233	Elonex	108c	4236
Elite	1019	4121	NCR	101a	4122
EPSON	1008	4104	Phoenix	100a	4106
Fujitsu	10d0	4304	Newbridge	10e3	4323
Future Domain	1036	4150	HITACHI	1037	4151
Gemlight	109b	4251	Megachips	109c	4252
Genoa	1047	4167	Fountain	1049	4169
Goldstar	107c	4220	Leadtek	107d	4221
Hitachi	1020	4128	Oki	1021	4129
Hyundai	106c	4204	Sequent	106d	4205
IBM	1014	4116	ICL	1016	4118
ICL	1056	4182	Motorola	1057	4183
Intergraph	1091	4241	Diamond	1092	4242
Interphase	107e	4222	Tulip	1085	4229
Matrox	102b	4139	Chips and	102c	4140

Manufacturer	Man. ID	PNP ID	Manufacturer	Man. ID	PNP ID
			Tech.		
Matsushita	10f7	4343	Altos	10f8	4344
Micronics	1012	4114	Cirrus Logic	1013	4115
Mitsubishi	1067	4199	Apple	106b	4203
National Instruments	1093	4243	Quantum Designs	1098	4248
National Semi	100b	4107	Tseng Labs	100c	4108
NCR	1000	4096	ULSI	1003	4099
Neomagic	10c8	4296	Fujitsu	10ca	4298
Networth	107a	4218	Gateway 2000	107b	4219
OAK	104e	4174	Hitachi	1054	4180
PC Direct	10f9	4345	Truevision	10fa	4346
Reply Group	1006	4102	Netframe	1007	4103
Rockwell	1112	4370	Zilog	1121	4385
S3	5333	21299	Intel	8086	32902
Samsung	1099	4249	Packard Bell	109a	4250
SGS Thomson	104a	4170	Buslogic	104b	4171
Siemens	1029	4137	LSI	102a	4138
Spea Software	1017	4119	UNISYS	1018	4120
Tandem	10e4	4324	Micro Industries	10e5	4325
Tatung	103b	4155	HP	103c	4156
TI	104c	4172	SONY	104d	4173
Toshiba	102f	4143	Miro Computer	1031	4145
Tsenglabs	10be	4286	Samsung	10c3	4291
Video Logic Ltd	1010	4112	Digital	1011	4113

Manufacturer	Man. ID	PNP ID	Manufacturer	Man. ID	PNP ID
Vitesse	101b	4123	Western Digital	101c	4124
VLSI	1004	4100	ALR	1005	4101
Vtech	105e	4190	United Micro	1060	4192
Wyse	102d	4141	Olivetti	102e	4142
Xilinx	10ee	4334	Creative	10f6	4342
Yamaha	1073	4211	Nexgen	1074	4212

H. References.

1. AVR microcontrollers from the website www.atmel.com

2. *PCI Local Bus Specifications*, *Revision 2.1.* PCI Special Interest Group, June 1995.

3. Shanley, Tom. *PCI System Architecture.* MindShare, Inc., 1995.

4. Website of PLX technology.

5. MCS9835CV website.

6. MosChip website.

7..NET – Complete Development Lifecycle Gunther Lenz, Thomas Moeller, Addison Wesley, 2003.

8. Peterson, Wade D. *The VMEbus Handbook.* VITA, 1989.

9. Solari, Edward. *ISA & EISA Theory and Operation.* Annabooks, 1992.

10. Theus, John. *FutureBus+ Coming of Age.* A three-part series on the FutureBus+ standard. MicroProcessor Report, 6(7-9), May-July 1992.

11. L. Sha, J. P. Lehoczky, and R. Rajkumar, “Real-Time Scheduling Support in Futurebus+,” in Proceedings of the Real-Time Systems Symposium (I. C. S. Press, ed.), (Lake Buena Vista, Florida, USA), p. 331, IEEE Computer Society Press, Dec. 1990.

12. IEEE, ”New York, NY, USA”, IEEE Standard 896.3-1993, IEEE recommended practice for FutureBus+, July 1994.

13. Tving, Ivan. *Multiprocessor interconnections using SCI.* ID-DTH, 1993.

14. Infiniband Trade Association, Infiniband Architecture, Mar. 2000.

15. International Organization for Standardization (ISO), CAN, International Standard 11898, 1993.

16. P. Pedreiras and L. Almeida, “A Practical Approach to EDF Scheduling on Controller Area Network,” in IEEE/IEE Real-Time Embedded Systems Workshop (Satellite of the IEEE Real-Time Systems Symposium), (London, UK), IEEE/IEE, Dec. 2001.

17. K. Zuberi and K. Shin, “Scheduling Messages on Controller Area Network for Real-Time CIM Applications,” IEEE Transactions On Robotics And Automation, vol. 13, pp. 310–314, Apr. 1997.

18. M. Hayter and D. McAuley, “The desk area network,” Tech. Rep. CS-TR-228, University of Cambridge Computer Laboratory, May 1991.

19. Universal Serial Bus 2 Specification, Apr. 2000.

20. Intel Corp., Accelerated Graphics Port Interface Specification. Santa Clara, May 1998. Revision 2.0.
21. Wang, Karl. et. al., *Designing the MPC105 PCI Bridge/MemoryController.* IEEE Micro, April 1995.
22. IEEE Std 1149.1-1990 (Includes IEEE Std 1149.1a-1993), IEEE StandardTest Access Port and Boundary-Scan Architecture, ISBN 1-55937-350-4, IEEE order number SH16626.
23. JTAG – ICE user manual available in AVR Studio Help file.
24. iButton overview http://www.maxim-ic.com/an3808.
25. Overview of 1 – wire technology www.maxim-ic.com/an1796

I. CD Contents

The CD enclosed contains the following data:

- Coding Folder, containing coding for examples in chapter 8 and Bootloader of PCI card.
- Hercules setup
- Datasheets
- Schematic diagram of PCI card (main11.schdoc)
- Thesis in PDF Format(Chirutkar.pdf)

J. Paper published and Conferences Attended.

1. **Chirutkar Harshadkumar G.** & Dr. H. N. Pandya**,**
   A versatile low cost PCI Bus Interface Card useful as AVR Microcontroller trainer system is accepted in IEEE International Conference on Process Automation, Control and Computing PACC2011 at **Coimbatore Institute of Technology,** Coimbatore, July 20-22, 2011.

2. **Chirutkar Harshadkumar G.** & Dr. H. N. Pandya,
   Secure Digital Access system using iButton in Construction section of Electronics for You magazine, November, 2010 issue.

3. **Chirutkar Harshadkumar G.** & Dr. H. N. Pandya,
   Study of Image Processing techniques using Image processing Toolbox of Matlab in Lab experiments September 2010, Page 306.

4. **Chirutkar Harshadkumar G.** & Dr. H. N. Pandya,
   AVR Hex File Manipulator in Software section of Electronics For You magazine, October, 2009 issue

5. **Chirutkar Harshadkumar G.** & Dr. H. N. Pandya,
   GSM Communication based Voltage Supervisor in the proceedings of XXXII National Systems Conference 2008 at Dept. of Electrical Engineering, IIT Roorkee, page 25.

6. **Chirutkar Harshadkumar G**. & Dr. H. N. Pandya,
   ATmega32L Microcontroller based Communication between PC and Sony IR Remote Control published in **Electronics Maker**, October2008 issue, page 46-49.

7. **Chirutkar Harshadkumar G**. & Dr. H. N. Pandya,
   Communication between PC and Infrared Remote Control Using ATmega32L Microcontroller presented at GUJCOST sponsored State Level Paper (Oral) Presentation **Science Excellence – 2008** at

Gujarat University, Ahmedabad on 5th January, 2008, and received 2nd prize.

8. **Chirutkar Harshadkumar G**. & Dr. H. N. Pandya,
   Atmel AVR ATtiny15L Interrupt Concepts, Programming and Utilization published in **Lab Experiments Journa**l, December – 2007 Issue.

9. **Chirutkar Harshadkumar G.**, Mr. A. A. Bhaskar, Dr. H. N. Pandya **Microcontroller based Serial Communicator** presented at UGC sponsored National Seminar **Emerging Trends and Developments in Embedded Systems** at D.M.'s College, Goa conducted by Department of Electronics, D.M.'s College, Goa on 13th and 14th March, 2007.


NARAYAN HARI
NARAYAN HARI
HGC0EHNP NARAYAN HARI HGC0EHNP
3.3V
5V
JTAG
MOSCHIP
TEST OK
NARAYAN HARI